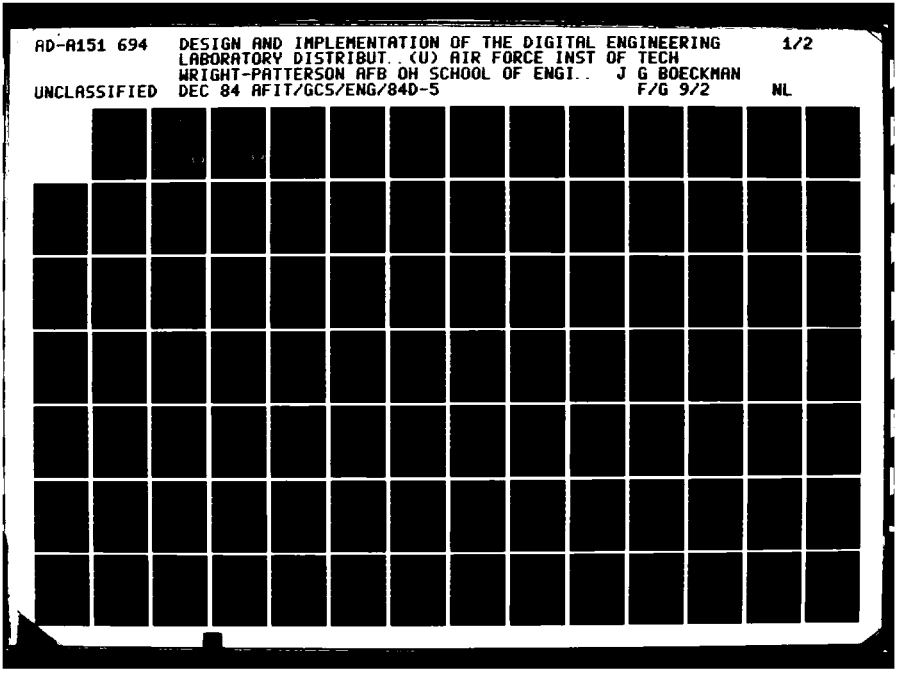
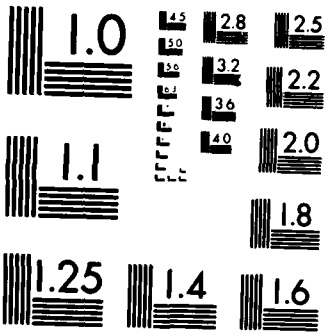

AD-A151 694
DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUT..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J G BOECKMAN
1/2
UNCLASSIFIED
DEC 84 AFIT/GCS/ENG/84D-5
F/G 9/2
NL



(1)

DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

John G. Boeckman
Captain, USAF

AFIT/GCS/ENG/84D-5

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

John G. Boeckman
Captain, USAF

AFIT/GCS/ENG/84D-5

Approved for public release; distribution unlimited

AFIT/GCS/ENG/84D-5

DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

John G. Boeckman, B. S.
Captain, USAF

December 1984

Preface

The purpose of this study was to design a distributed database management system (DDBMS) for use in the AFIT Digital Engineering Laboratory (DEL). First, a requirements analysis was accomplished using Structured Analysis and Design Technique (SADT) diagrams. Next, a detailed design was done finishing up the SADT diagrams and using structure charts to show the module decomposition. Part of the design was implemented using computers in the DEL computer network. The results of the implementation were analyzed and proposals were made for future DDBMS projects.

In performing the design and implementation, and the writing of this thesis, I have received a great deal of help from others. I am deeply indebted to my thesis advisor, Dr. Thomas C. Hartrum, for his skillful help in time of need. I also wish to thank Dr. Henry Potoczny and Major Walter Seward, who were also on my thesis committee, for their time and effort. Many thanks to Dan Zambon and Charlie Powers for their technical assistance in the lab. Finally, I wish to thank my dear wife Alice for her continuous, loving support throughout this thesis, and for her many hours of filling in SADT diagrams and structure charts.

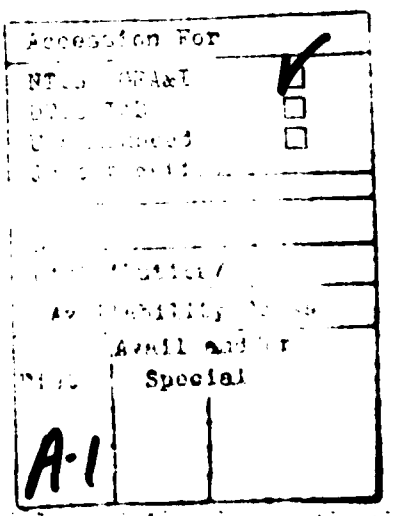


Table of Contents

The following additional thesis volumes are maintained at AFIT/ENG:

Volume II: DDBMS Requirements Definition

Volume III: DDBMS Detailed Design

Volume IV: DDBMS Partial Implementation Design

List of Figures

Abstract

This effort produced a basic design and partial implementation of a distributed database management system (DDBMS) for use in the AFIT Digital Engineering Laboratory. The objectives of this thesis were to lay out the requirements for a DDBMS, to design a simplified implementation of one, and to accomplish a partial implementation of that design.

The requirements analysis used the Structured Analysis and Design Technique (SADT) to document the DDBMS requirements. The analysis, independent of any hardware or specific algorithmic implementation, covers all aspects of a DDBMS, including routing and execution of queries and updates, DDBMS initialization and reconfiguration, and recovery from network malfunctions.

The detailed design expanded the SADT diagrams of the requirements analysis for specific methods of executing queries and updates in the DDBMS. These methods were selected to limit the scope of that design. Structure charts were produced using the SADT diagrams as a reference, specifying parameters and algorithms for the modules.

Only the part of the design that handles DDBMS queries was implemented. The implementation was greatly simplified to exclude query partitioning and optimization. Two DDBMS

nodes were connected via the Digital Engineering Laboratory network. These were, in turn, connected to host computers that evaluate queries and return the resulting relation.

DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

I. INTRODUCTION

Background

A database is a large collection of data that is organized for rapid retrieval and updating. Software that controls data manipulation and structure in a database is known as a database management system (DBMS). Databases that reside on a single computer are known as centralized databases, while databases that are spread over several computers in a network are known as distributed databases. The software that manages tasks occurring in a distributed database is a distributed database management system (DDBMS). Other terms used frequently in this thesis are defined in a glossary in Appendix A.

There are several advantages that a distributed database has over one that is centralized. Users have access to larger amounts of data. More users can access the data. If one computer goes down in the network, other computers can still operate in the network without it. Data can be spread over several computers, lowering access time and storage requirements for each computer.

Naturally, there are also disadvantages. Complexity is

increased dramatically in a distributed database system. Most distributed databases in use today are custom-designed for a specific application. Many data problems must be overcome in a distributed database; for example, keeping copies of the same data concurrent, preventing data deadlock, and running queries and updates efficiently. Keeping track of data on several computers requires an extensive data dictionary spread among the computers in the network.

There are three approaches to distributed database management systems. The integrated, homogeneous, and heterogeneous models are shown in Figure 1 (5:7-10). In the integrated model, each DBMS is designed being connected to the others in the network, and can access them without data translation, as shown in Figure 1a. This strategy reduces the useful CPU time at each computer, and requires memory to store the data exchange process, two reasons for its lack of popularity.

The homogeneous model removes the network data exchange module from memory at each computer, and installs a communication software module between each computer and the rest of the network, as shown in Figure 1b. In this model each computer must support the same DDBMS as the others. This is the most common method of implementing distributed databases.

The heterogeneous model can link different DBMSs

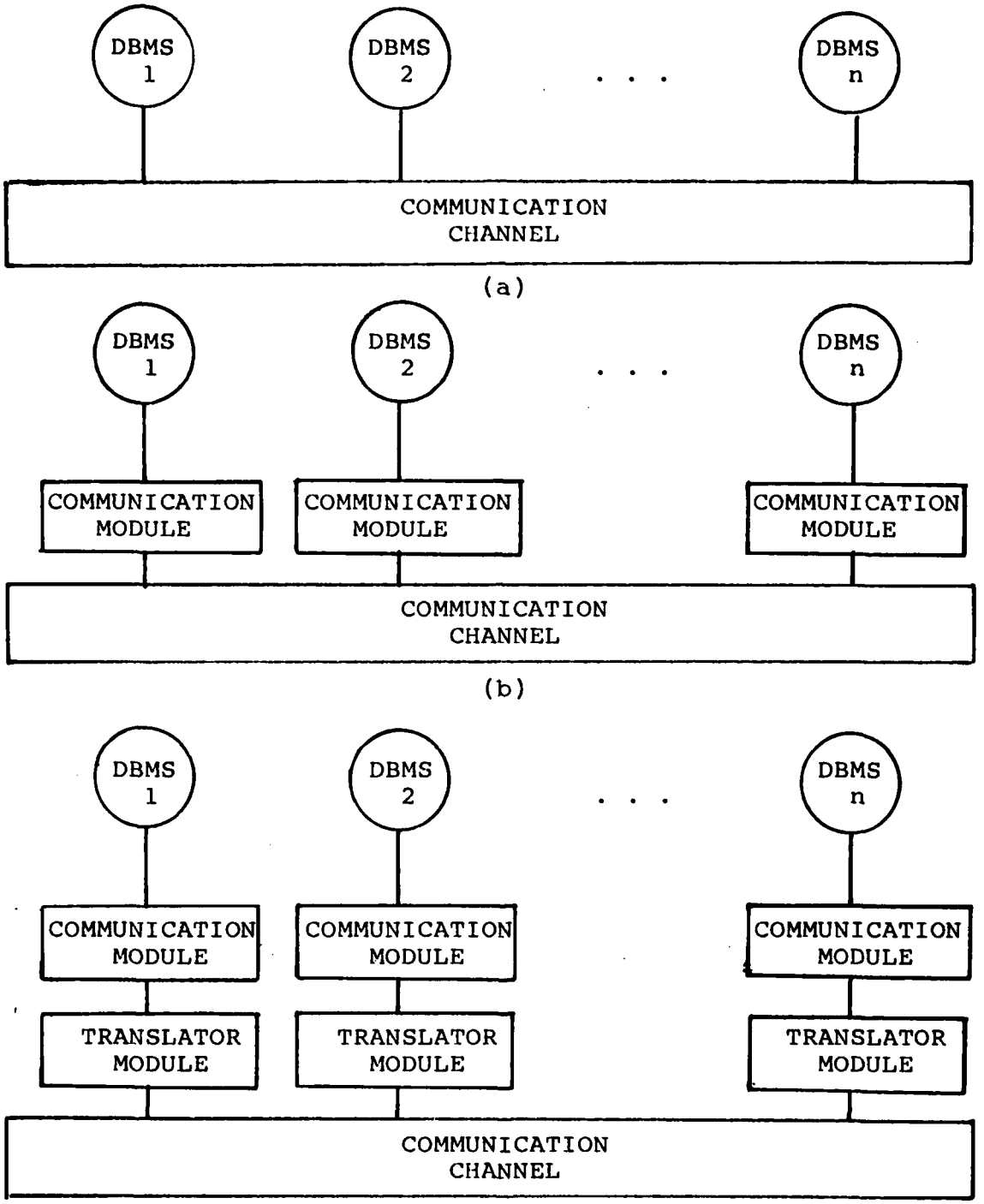


Figure 1 (a) Integrated Architecture
(b) Homogeneous Architecture
(c) Heterogeneous Architecture (5:10)

together through a distributed database. Since the DBMSs are incompatible, a translator software module is installed between the communication module and the network communication channel, as in Figure 1c. This is the most flexible model, but also the most complicated to implement. A universal database model would greatly enhance the usability of a heterogeneous model, since software could be written to translate from any type of DBMS language to the universal model, and from the universal back to the original language. Also, all network functions in the DDBMS could be written in that one universal language.

Air Force Applications

The Air Force is currently engaged in large-scale research and development to harness the technology of distributed databases to solve Air Force problems. Dawson of Mitre Corporation (2) discusses using distributed databases for a field-deployable, tactical air control system.

The Worldwide Military Command and Control System is heavily dependent on networking capabilities, and in an article Coles of Mitre Corporation (1) discusses current data management and distributed processing problems. Later, the article discusses computer architectures and design approaches, alternatives to current methods of operation, that could lead to improved future systems.

Rome Air Development Center (RADC), Griffis AFB, NY,

sponsor of this thesis, is doing distributed database research through contracts with the Computer Corporation of America which published a three-volume paper describing their efforts (6, 7, 8). In the first volume, the contractor describes the foundation necessary for understanding concurrency control and recovery from network problems (6). The second volume describes work on the performance analysis of concurrency control algorithms (7). The final volume provides a handbook of information about a number of important concurrency control algorithms used in the design of a distributed database management system (8).

Another Air Force organization involved in distributed database processing is the Space Defense Operation Center (SPADOC) in Colorado Springs, Colorado. TRW has developed a model of the SPADOC data processing environment (9) that maintains tables on database access requirements, file locations, and other performance data. The model is used to fine-tune performance by measuring and analyzing the design effectiveness of the distribute database at SPADOC.

Summary of Current Knowledge

The DDBMS is executed in a network with a group of host computers connected via low bandwidth communication lines. The network may be either a local area network with the computers in close proximity, or a network spread across geographically-dispersed locations. The network appears as

in Figure 2, with each site connected in some communication configuration in the network. One of the sites in the network is chosen at startup time as the central site, and contains the Centralized Network Data Directory (CNDD), which stores locations of all data items in the distributed database (5:69). The central site also stores files of pending updates, which contain updates to inactive sites in the network which have copied data that was updated at an active site.

At each site, a software module executes to perform that particular site's functions in the distributed database. That software module must interface directly with the user, the local DBMS, the network, and the network data directories at the site. The functions of the network data directories will be described in the following sections. Figure 3 gives an overview of the distributed database interface design. The design leaves open the option of having either a multiprogramming system, such as a VAX 11/780 computer, or a dedicated processor system, such as an LSI-11, as sites in the distributed database. For a VAX-type system, the DDBMS software would be one of a number of processes, including the DBMS interface to the distributed database, running concurrently in memory. For an LSI-type system, the DDBMS would run as a dedicated process communicating with the local DBMS in an attached host computer.

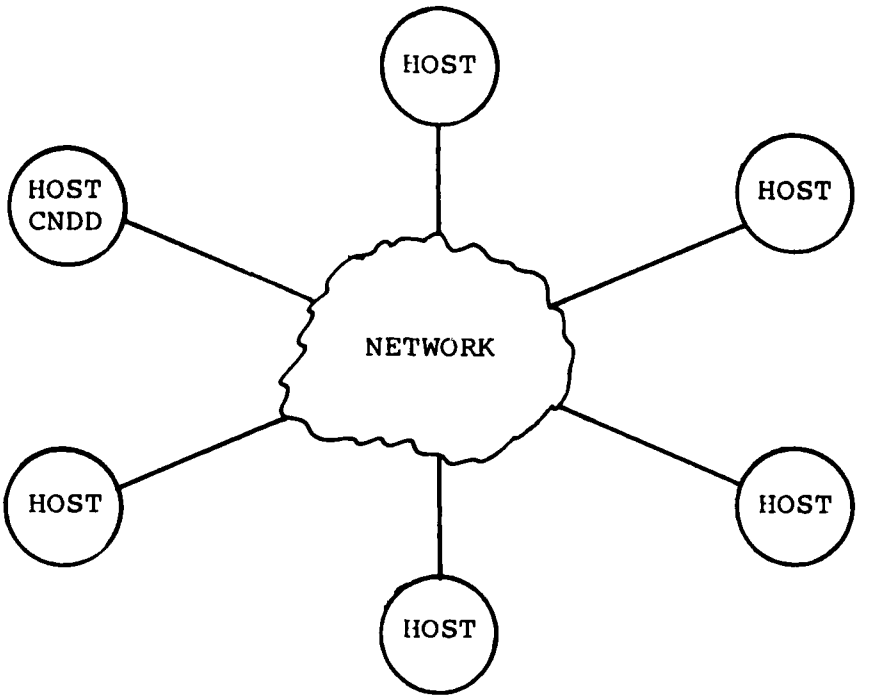


Figure 2 Network Structure in the DDBMS

In 1982, Capt Eric F. Imker (5:63-79) produced a high-level design of a distributed database management system (DDBMS) for use on computers in the AFIT Digital Engineering Laboratory (DEL). Though the structure of his design is not used in this thesis, it is presented here to maintain its consistency. His design includes three parts:

(1) The Network Access Process (NAP) which includes communications hardware and software to link each computer to the rest of the network, and also information on the status of the rest of the DBMSs in the distributed database.

(2) The Network Data Directory (NDD) which maintains information on the location of data in the distributed database.

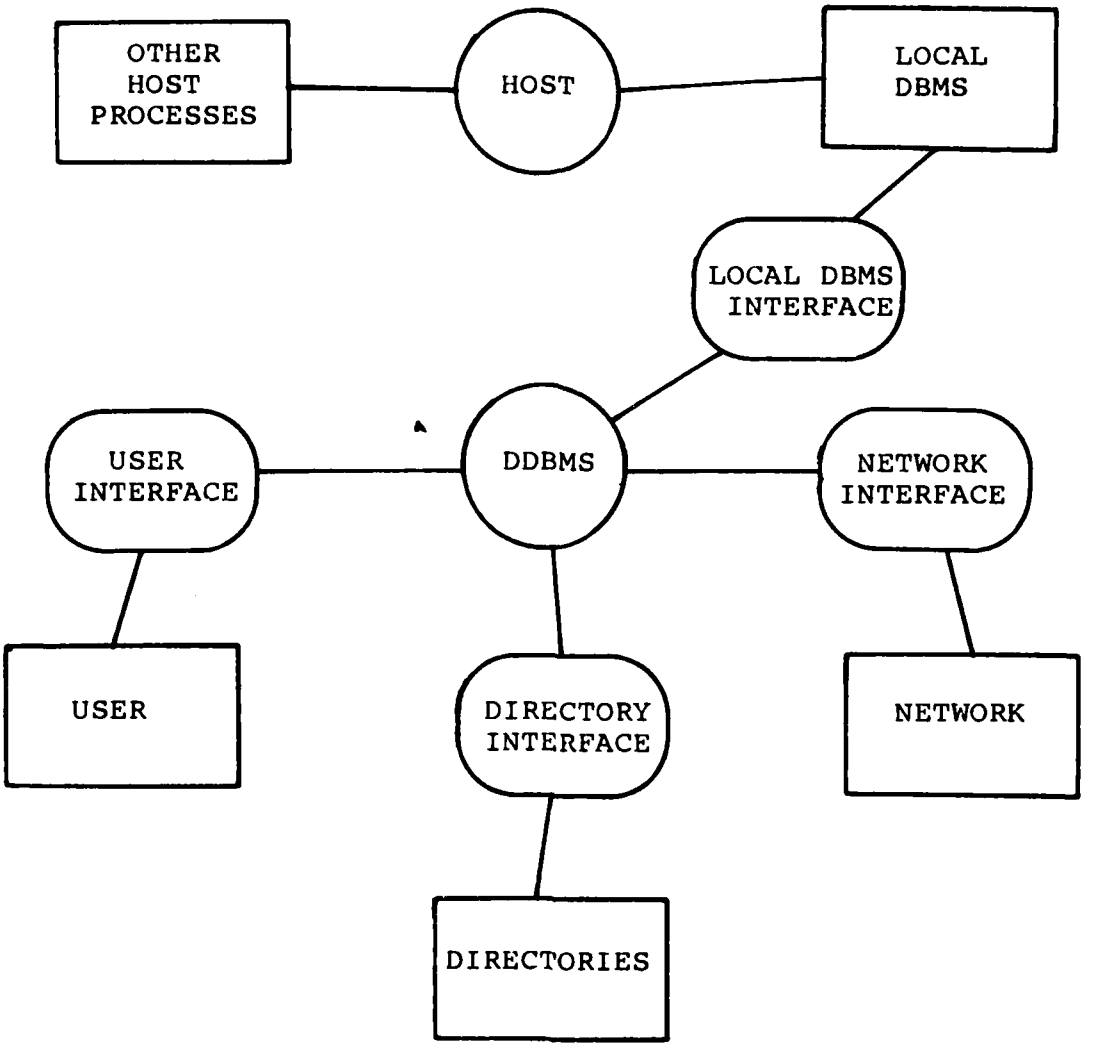


Figure 3 Interfaces in the DDBMS

(3) The Network Database Management System (NDBMS) which provides an interface between each local DBMS and the entire distributed database.

Network Access Process

The Network Access Process (NAP) is the term used by

Imker to describe the function of interfacing the DDBMS at each site with the network. The Network Operating System (NETOS) in the DEL network handles the routine functions of the network, sending and receiving messages, protocols, etc. The NETOS functions under a seven-layer protocol governed by the Reference Model of Open Systems Interconnections (OSI) developed by the International Standards Organization (ISO). (reference the LSINET Network Protocol) The model outlines the framework of a protocol scheme for computer system communications and interfaces. Figure 4 illustrates the seven ISO layers (11:453-487).

Each layer is connected via subroutine calls and hardware interfaces only to the layer above and the layer below. The Physical Layer, the first layer, is concerned with primarily the hardware connection protocol. The second layer, the Data Link Layer, groups the data into data frames and checks for correct transmission of the frames. The Network Layer, layer three, manages the routing of data frames around the network. Layer four, the Transport Layer, is used to establish host-to-host communication, to transmit variable-length buffers of data, and to terminate the host-to-host link. The fifth layer, the Session Layer, converts between logical and physical source/destination references. Layer six is known as the Presentation Layer and transforms application program inputs into standard NETOS formats. Finally, layer seven consists of the Application Layer,

which involves a particular application running on a host computer in the network.

Figure 4 shows that the DDBMS software interfaces with the NETOS at layer five. This has been chosen since the DDBMS software provides only logical and not physical source/destination references, and the software handles the functions of the Presentation Layer, layer six, by preparing inputs in the standard NETOS format. Imker also conceived the NAP maintaining a table of status indicators for the other sites in the distributed database.

Network Data Directory

The Network Data Directory (NDD) consists of the Local Network Data Directory, the Centralized Network Data Directory, and the Extended Centralized Network Data Directory. The Local Network Data Directory (LNDD) contains data locations for all data items residing at a particular location. The NDBMS provides updates to the LNDD.

The Centralized Network Data Directory (CNDD) contains directory information about each remote DBMS. There is only one of these in the distributed database. It maintains the data entity names and location where each data entity exists. The NDBMS will also provide updates to the CNDD.

The Extended Centralized Network Data Directory (ECNDD) is a smaller version of the CNDD, and one is located at each site in the distributed database. The ECNDD contains a data

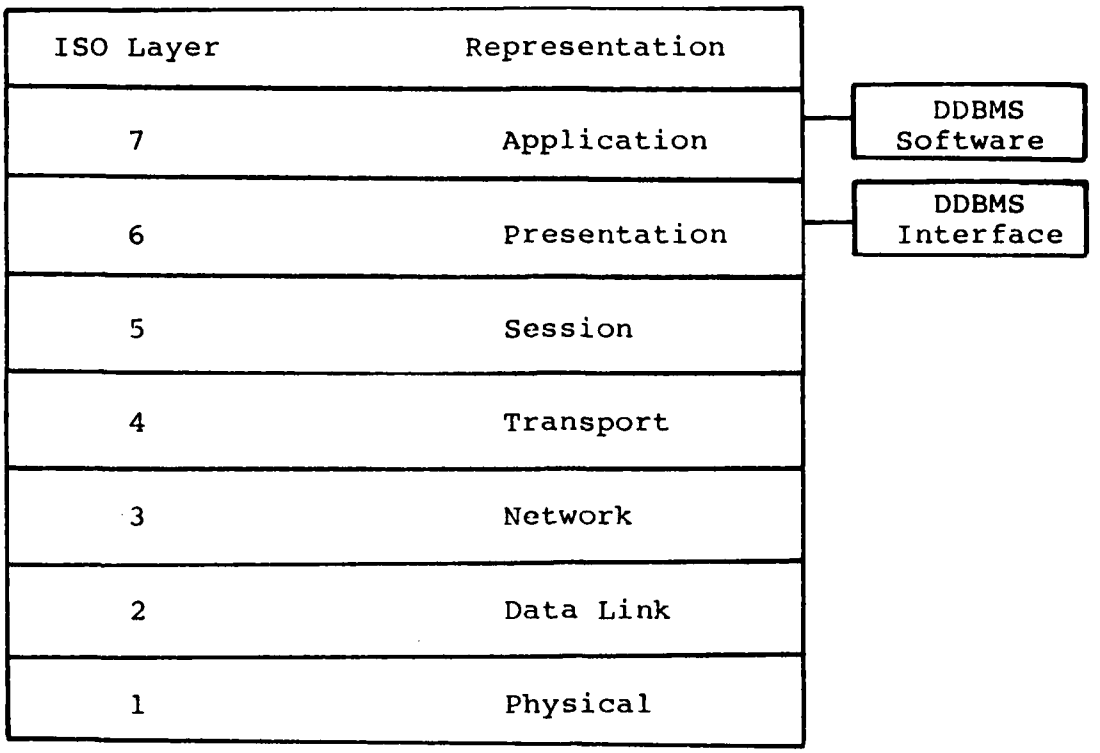


Figure 4 ISO Layer-DDBMS Interface

entry location for each data item that is requested by a particular site. Since the ECNDD could conceivably grow to the size of the CNDD, it will be limited to "N" items, where N is an experimentally determined value. The NDBMS will also provide the ECNDD with updates, which will replace items in the ECNDD by using a Least Recently Used (LRU) algorithm.

Network Database Management System

The Network DBMS (NDBMS) is the term used to describe the main software module of the DDBMS at each site in the network. It provides an interface between a local DBMS and

the distributed database. It consists of the interfaces to the user, the local DBMS, the NAP, and the LNDD; a query processing procedure; a concurrency control process; and backup and recovery procedures.

Interfaces

The NDBMS interfaces (to the user, the local DBMS, the NAP, and the LNDD) allow the NDBMS a means to communicate to each of these areas. The NDBMS uses these interfaces to coordinate its activities with the other sites in the distributed database.

Query Processing Procedure

The query processing procedure consists of a universal data model translator, a replication process using the ECNDD at a site, a universal query language translator, and a query optimization process. The universal data model translator transforms query results from data entities in the format of the local DBMS into universal data entities.

The replication process maintains a table to indicate which remote data entities are used and how often. This table, stored on the ECNDD, is needed to determine how to replicate data based on the amount of usage. In the table there is a counter for each data entity in the ECNDD. If the counter reaches "M" (an experimentally determined value) before the entity is purged then the entity is replicated at the local computer, if possible. Replicating an entity

includes removing the item from the ECNDD, adding the item to the LNDD, and informing the CNDD of the replication.

The universal query language translator, which retrieves remote data, builds a query to forward to remote computers when the NDBMS requires data from those computers. Each query is decomposed at the computers with the data. Then the resulting data is returned to the originating computer for further processing.

Also, the NDBMS optimizes each query function. Queries requiring remote data are divided into two types, unique and replicated requests. Unique requests are those for which all the data entities desired exist on one computer. For these, the NDBMS forwards the request to a computer that can most efficiently process the request. At that computer, the request is processed completely, and the result is returned to the original computer. Replicated requests consist of local and remote or multiple remote requests. For these, a recursive technique developed by Eugene Wong (14:50-68) will be used in future thesis efforts to optimize query processing by minimizing data movement.

Concurrency Control Process

The concurrency control process ensures that copies of replicated data are updated concurrently with one another. Imker chose the majority consensus voting algorithm to accomplish concurrency since it forces copies of replicated data to converge to the same values, and it also prevents

deadlock. In the algorithm, individual DBMSs with copies of the data vote on an update request. If a majority vote to accept the request, then it is applied to all copies of the data. DBMSs that vote use timestamps to check the validity of an update.

There are six steps used in the voting and updating procedure:

(1) A DBMS receives a request and decomposes it to obtain the data elements required for the update and the timestamps associated with each data element.

(2) The DBMS creates a modified update request by attaching the new values of the data elements and their respective timestamps.

(3) The request is forwarded to all other DBMSs which have a copy of the data.

(4) Each DBMS with a copy of the data votes on the update.

(5) If the request is approved by a majority of the DBMSs which are involved, then the update is made.

(6) If the request is rejected, then the originating DBMS may resubmit the request.

Statement of the Problem

The objective of this thesis was to continue Imker's work in the design of a DDBMS. To complete the design, this project produced a detailed requirements analysis and a

detailed design of the DDBMS software required at a typical node (computer in a network). Also, software to connect two nodes in the distributed database was written to test the DDBMS design.

The main purpose of a distributed database is to allow the user to access several databases spread over a network as if they were one. This is accomplished through DDBMS software which optimizes a user's query to obtain the query in the least amount of time possible. Another purpose of a distributed database is to update copies of data in such a way that the copies converge to the same values (though copies of data might not have the same values at a particular instant in time). Deadlock, a state where one process is waiting for data held by the other process, must be prevented, or at least recovered from, to allow for continuous data throughput.

Scope

The detailed requirements analysis examined Imker's high-level design and developed a set of requirements stating what software modules are necessary for a DDBMS in the AFIT DEL. The basic setup for a DDBMS was made, though some complicated parts were simplified due to a lack of time. The universal data model, which will translate between different types of database management systems, was being designed concurrently as another thesis, making it not available for use. Instead, a relational structure was used

for intersite communications. Relational queries and data were passed over the network. Two relational translators were built to test the DDBMS software on various computers in the DEL.

The first six ISO layers were in place, so this project interfaced at the fifth layer without being involved in the deeper levels of the NETOS ISO structure. Only two sites were implemented to work out the many initial problems in the software. Queries were not optimized as in Wong's design, but sent either to a single site that contained unique data, or to the first site in a list of sites that had replicated data. The passing of queries and data over the network also was not optimized. An already in-place relational optimizer was used to optimize the queries concerning their relational structure.

No concurrency control or deadlock prevention algorithm was implemented. Updates for data replicated at two or more sites in the network were sent to the sites, and the sending site waited for the results. No specific deadlock prevention algorithm was implemented due to the small size and limited nature of the implemented distributed database.

Assumptions

The basic assumption in the design was the existence of a universal data model, which was needed to convert internode messages from one type of DBMS to another. At the

start of this thesis none existed, but a localized model was designed in another concurrent thesis. The model must provide the software link to allow different types of databases to communicate with one another, and is essential in developing a working DDBMS in the AFIT DEL.

Another assumption was the existence and functioning of the proper ISO layers needed for communications in the network. This was necessary to be able to pass messages and data between sites in the network. Also, it was assumed that the network has links to computers with different types of relational DBMSs available.

Approach

The thesis effort was accomplished in the following four sequential steps:

1. Requirements Analysis
2. Detailed Design
3. Partial Implementation
4. Analysis of Implementation

The requirements analysis is a description using SADT diagrams and descriptions of all the software components necessary to produce a working DDBMS at the AFIT DEL. The analysis was meant to be as general as possible, usable for any organization needing to set up a DDBMS. The analysis is decomposed up to the point where implementation decisions had to be made for this thesis.

The detailed design finishes out the SADTs of the

requirements analysis, spelling out implementation decisions made to limit the scope of this thesis, and uses structure charts and pseudocode to state how to implement those requirements. The detailed design includes the simplifications discussed in the Scope section.

The partial implementation placed the DDBMS software on two communicating nodes in the NETOS system, and implemented the code derived from the detailed design. The two nodes had different types of DBMSs to test out the ability to translate between different types of relational DBMSs.

The analysis of implementation evaluated the advantages and disadvantages of the partial DDBMS implementation. Studies were made of response times, accuracy of updating copied data, accuracy of data returned from queries, occurrances of deadlocks, and ways to overcome these problems through future thesis efforts in these areas.

Overview of the Thesis

The format of this thesis follows the approach taken during the project. Chapter II presents a description of the Requirements Analysis completed on the necessary parts of a DDBMS using SADT. Chapter III presents the detailed design of a DDBMS for use in the AFIT DEL. Chapter IV describes the methods of coding and testing used in implementing part of the DDBMS. Chapter V analyzes the effectiveness of the requirements analysis, design, and

implementation parts of the thesis. Chapter VI summarizes the thesis and recommends action to be taken for future thesis projects.

II. REQUIREMENTS ANALYSIS

Introduction

The Requirements Analysis section of this thesis was accomplished using the Structured Analysis and Design Technique (SADT) (9:62-64). Appendix B lists the requirements by number and Volume II of the thesis contains the SADT Requirements Analysis design. The Requirements Analysis is set up to be independent of the number of sites in the DDBMS, the database languages of the individual sites, the configurations of the network utilized in the DDBMS, and the hardware characteristics of the individual sites. First is an explanation of the basic software requirements for each site in the DDBMS, which is followed by the decomposition of the requirements.

Basic Requirements

Appendix B specifies the general requirements of the DDBMS software. The required functions fall in six categories:

(1) Initialize the DDBMS.

(2) Maintain status information on other sites.

(3) Reconfigure the DDBMS.

(4) Transmit and receive messages and data to other sites and to the host computer.

(5) Update and maintain the ECNDD and LNDD.

(6) Execute queries and updates. If chosen as the

central site, executes CNDD and pending update functions.

Initialization of the DDBMS occurs at startup time. There are two possible conditions: either the site in question is chosen as the central site, or it is not. If it is chosen as the central site, it initializes CNDD and database data at the site, sends status query messages to the other sites, receives the responses from the sites, and begins execution of the DDBMS. If the site is not chosen as the central site, it initializes that site's database data and returns the central site's initial message when it receives it.

Each site maintains a status information table on every other site in the DDBMS. This is necessary for broadcast-type messages, and also for obtaining remote data or sending replicated updates. These tables are maintained through messages about DDBMS malfunctions and reconfigurations passed between sites.

The DDBMS can be reconfigured in the following ways. A site can be added to the DDBMS, which involves notifying the central site and all other sites of the new site. Also, the data at the added site is updated through pending updates maintained at the central site. A site can also be deleted, involving notification of all sites and the beginning of a pending update file for that site at the central site. The CNDD and pending update files can be copied from one site to another, designating the new site as the central site, and,

again, all sites would have to be notified of the transaction. Finally, a mishap could occur in the DDBMS, including the crash of the central site, the crash of some other site, or some inactive network lines, and recovery would have to take place along with notifying all sites as to what measures were taken.

Messages and data must be transmitted and received at each site in the DDBMS, across the network and to each host computer, for it to function properly. The network operating system is in charge of these functions, though routines are needed to prepare the messages and data for transfer.

The three network data directories must be kept current about the locations of data in the DDBMS. The Local Network Data Directory (LNDD) contains locations of that site's data items and indicators for which items are copied. The Centralized Network Data Directory (CNDD) contains locations for all DDBMS data items, and exists only at the central site. The Extended Centralized Network Data Directory (ECNDD) exists at each site and contains locations of remote data items previously retrieved from the CNDD.

Each site has software to handle locally- and remotely-originating queries and updates. Also, if chosen the central site, it must handle the functions of updating the CNDD, retrieving locations from the CNDD, and updating pending update files.

Decomposition of the Requirements

Figure 5 shows the three overall functions in this DDBMS analysis, to initialize the DDBMS at startup time, to reconfigure the DDBMS in the event of a site crash or an external command from an operator, and to execute the DDBMS at the individual sites. The figure is taken from node A0 of the SADT section of Volume II. Each of these functions will be explained in the following sections.

Initialize DDBMS

"Initialize DDBMS" prepares the individual sites in the DDBMS for execution at the DDBMS startup time. This includes choosing a site to hold the Centralized Network Data Directory (CNDD), needed to indicate the site locations of the individual data items in the DDBMS. The site with the CNDD is known as the central site, or the CNDD site. Also, this routine will prepare other sites for execution in the DDBMS.

The routine initializes the central site by activating the CNDD there, determining which sites will participate in the DDBMS, and, once the initialization is complete, issuing a ready command to begin execution. Other sites in the DDBMS are initialized by a command sent to that site and a contact message from the central site, indicating that the site will participate in the DDBMS.

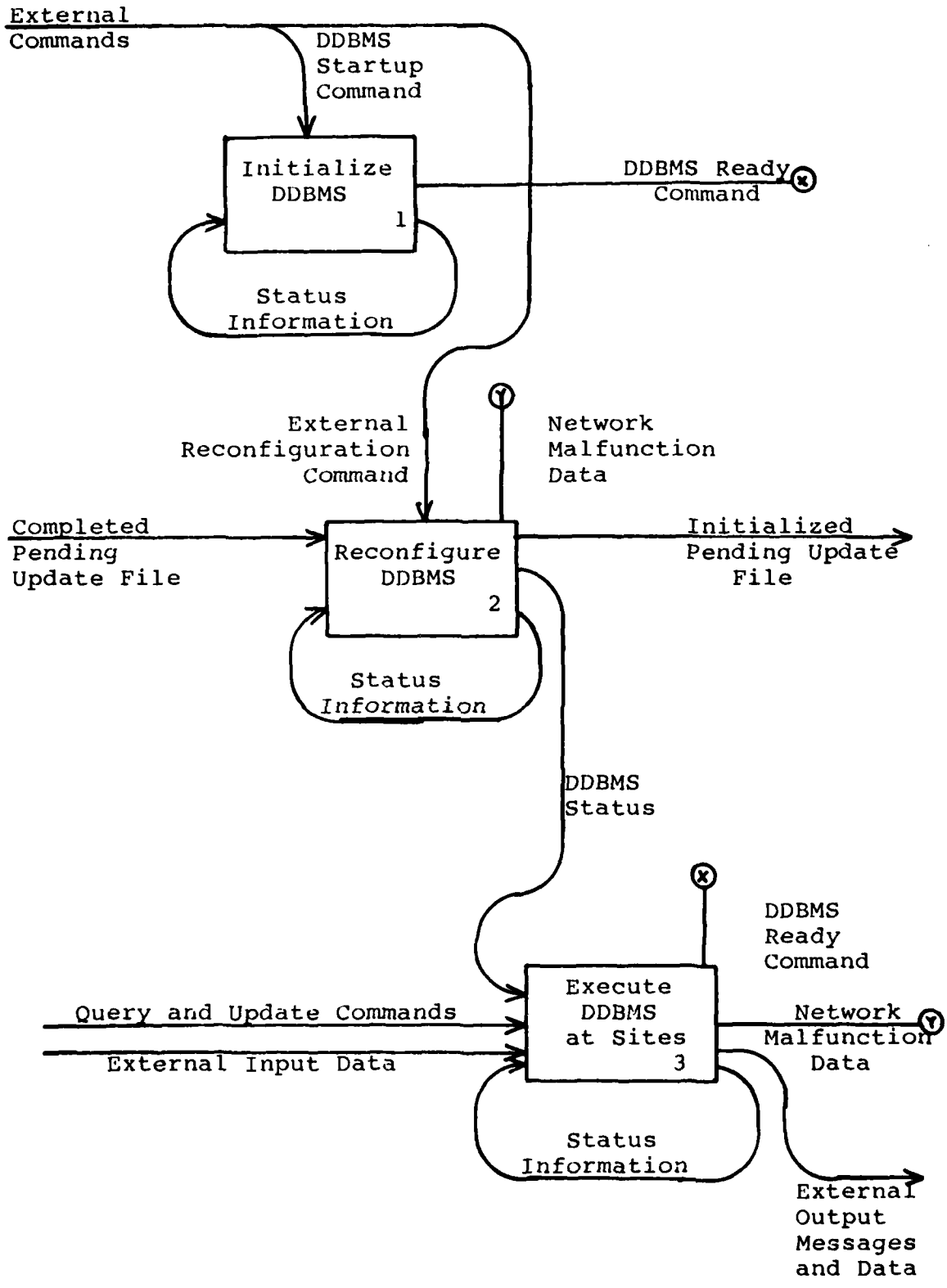


Figure 5 SADT Activity "Execute the DDBMS"

Reconfigure DDBMS

"Reconfigure DDBMS" changes the configuration of the DDBMS during execution. The DDBMS can be reconfigured either through an operator command, or by data about a malfunction in the network. Through a command, an operator may add a non-CNDD site to the DDBMS, delete a non-CNDD site from the DDBMS, or change the location of the CNDD from one site to another.

To add a non-CNDD site to the DDBMS, first communications must be established with the CNDD. Note that the location of the CNDD is passed to the site through the command to add a site from the DDBMS operator. The central site receives the new site's contact message, updates its list of available sites in the DDBMS, and sends an acknowledgement message back to the added site. The added site uses information about network conditions passed from the CNDD site to update its status information. Once this is complete, the added site then updates its data with a pending update file that contains updates to copies of the added site's data items transacted at other sites in the DDBMS while the added site was off-line. The pending update file is sent over the network from the central site. Finally, a message is sent to all sites to mark the added site active in their status information tables.

To delete a site from the DDBMS the deleted site sends a message to all sites to mark the deleted site inactive in

their status information tables. The CNDD site begins a pending update file of updates to data at active sites that is replicated at the deleted site. The pending update file is used to update the deleted site when it becomes active again. Relocating the CNDD moves the CNDD from one site to another. To accomplish this, the CNDD must be established at the new site. Then the new site will send a command to the old site to begin the copying process. All CNDD site data, including the pending update files, are sent to the new site over the network. When the copying process is complete, the status information is updated at both sites, to indicate the new location of the DDBMS. Also a message is sent to other sites in the DDBMS to notify them of the new CNDD location. Note that no transactions may occur against the CNDD or the pending update files during this relocating process.

Unlike the three previous operations, recovering from a network malfunction does not require an external command to occur. Controlled by a network mishap, it interprets CNDD and non-CNDD site crashes and down network lines, and does what is necessary to recover from them. If the CNDD site crashes, the CNDD is rebuilt at another site using the available LNDDs for input. If any site crashes, a pending update file is started for that site. If network lines are down restricting communication between sites, the network will be reconfigured so that sites will be able to

communicate properly with each other. Finally, messages are sent to the sites notifying them of the changes in the DDBMS that have taken place in this routine to recover from the network malfunctions.

Executing the DDBMS at the Individual Sites

Executing the DDBMS at the individual sites involves processing all site DDBMS functions other than resolving malfunctions and reconfiguring the network in the DDBMS. The SADT description defines the operation at the site that was selected to be the CNDD site, though only one site can hold the CNDD at a time. The overall parts of executing the DDBMS include receiving and transmitting messages and data from the network lines, updating the status information at a site about the current network configuration, interpreting input network messages, and processing the interpreted network messages at the site.

The network operating system controls transmitting and receiving messages and data over the network. An incoming message is read from the network and translated into usable form by the network operating system. Also, an outgoing message is packaged properly for transmission over the network. Data is decomposed into blocks which are individually transmitted over the network.

Input messages are determined to be either DDBMS status messages or network messages and local requests. DDBMS

status messages are used to update a site's information on the status of other sites, whether each site is active or inactive. Network messages and local requests are sent to a routine "Service Network Messages and Local Requests", which will be described in the following sections.

Service Network Messages and Local Requests

Figure 6 shows the decomposition of the routine. The types of messages that can be sent to this site are ECNDD updates, LNDD updates, or regular site requests. The ECNDD updates are parts of the ECNDD that are copied from changed parts of the CNDD. "Update ECNDD from CNDD Updates" uses these updates to keep the ECNDD current with the CNDD. "Update and Maintain LNDD" changes the LNDD to keep it current with the local database format that needs to be reflected in the DDBMS. The site sends a message to the CNDD site to update its information about the changed data item, and only after the site receives an acknowledgement from the CNDD site, stating that the CNDD is properly updated, will the local site update the LNDD.

"Service Requests" responds to user query and update requests by this and other sites in the DDBMS, and also determines network malfunctions and their causes. Its function will be more fully described in the next section. Once a network malfunction is determined by the "Service Requests" activity, it is deciphered using data from the available network malfunctions. If it is a non-CNDD site

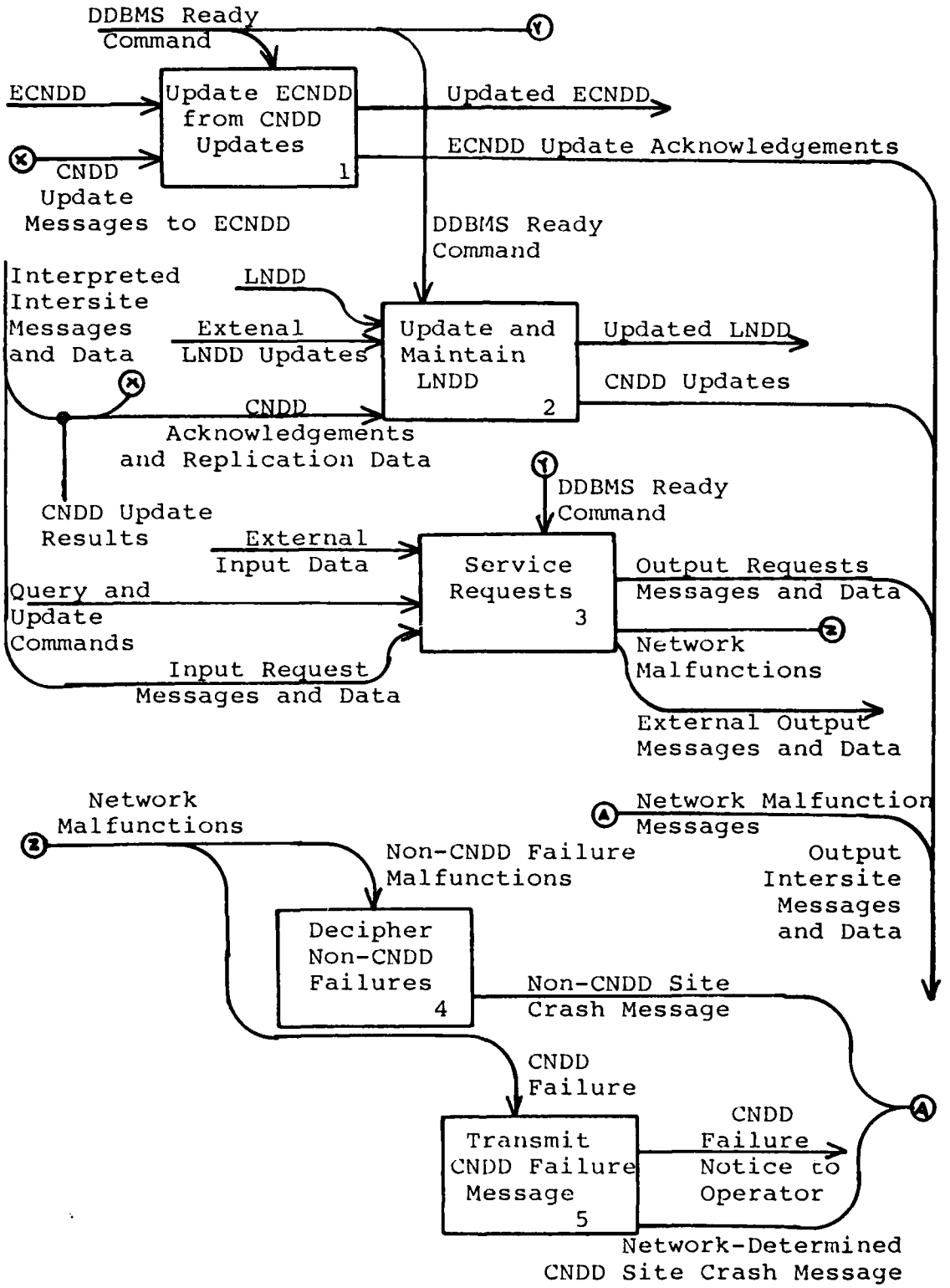


Figure 6 SADT Activity
"Service Network Messages and Local Requests"

crash, the network malfunction data is passed to the routine "Decipher Non-CNDD Failures" at the CNDD site to activate recovery from the malfunctions, a process described earlier. If it is a CNDD site crash, the malfunction data is passed to "Transmit CNDD Failure Message" which sends CNDD failure messages to the operator at the site and the next potential CNDD site in line. Then the operator determines if the CNDD is to be rebuilt at all, and if it is to be rebuilt whether it is rebuilt at the next site in line to become the CNDD site or a site of the operator's choosing.

Service Requests

Figure 7 shows the routines to handle the three types of requests that can be made to a DDBMS site: local requests, remote requests, and CNDD requests if the site happens to be the CNDD site. In this design, the CNDD is located at the site decomposed in this analysis. Were it not the CNDD site, the CNDD and pending update software would be turned off. Each of these types of requests are executed activities which define their functions.

"Service Local Requests" transacts query and update requests made at the local site. The routine first determines if the local request is a query or an update. Execution of local queries and updates are explained in later sections.

"Service Remote Requests" transacts query and updated

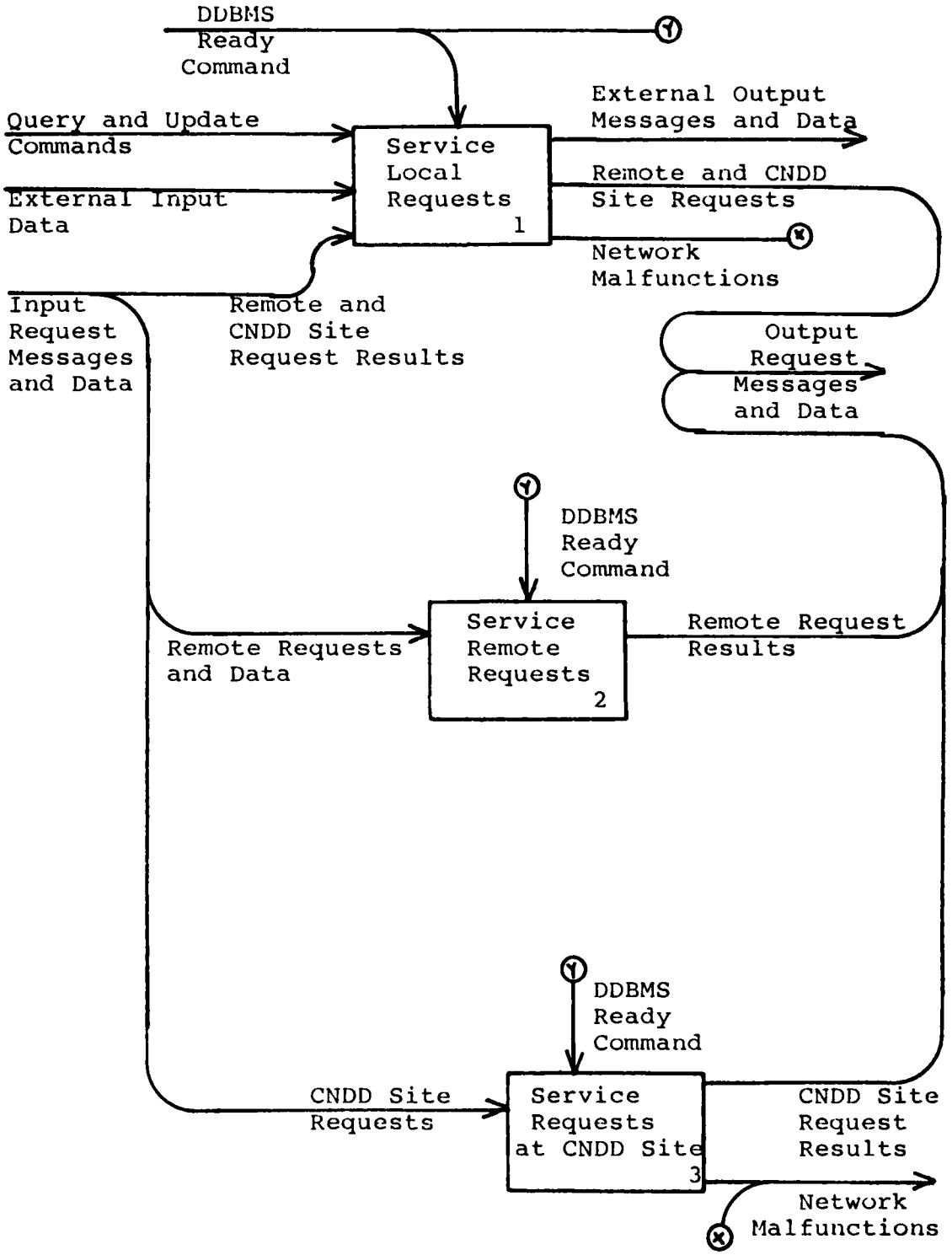


Figure 7 SADT Activity "Service Requests"

requests made by other sites in the DDBMS to the local site. The routine first determines if the remote request is a query or an update. Execution of remote queries and updates are explained in later sections.

"Service CNDD Site Requests" handles CNDD data location requests, CNDD updates, and pending update requests, that are made to the site in the DDBMS where the CNDD is located. Execution of these requests are also explained in a later section.

Servicing Local Queries

Local queries are queries that originate at the local site. The routine to execute them first determines if the local query needs only local host data to complete its task, a host query, or if it requires data from other sites in the network, a network query. Execution of host and network queries is explained in the following paragraphs.

To service a host query, first the query is translated from the universal database language into the local database language, if necessary. Next the host query is sent to the host computer for evaluation. There, the host computer evaluates the query, using the host DBMS, and returns the results to the DDBMS software. Lastly, the query results are translated, if necessary, from the local data model to the universal data model and sent to the user.

To service a network query, the query is first translated, if necessary, from the local database language

to the universal database language. Next, data locations are searched for in the ECNDD and the LNDD, and, if not found, then messages are sent to the CNDD site requesting locations from the CNDD. Once the locations are determined, the query is optimized according to its structure and the DDBMS configuration to allow it to execute as quickly as possible. The optimization function is not decomposed in the requirements analysis due to its dependence on a DDBMS query optimization algorithm. After the query is optimized, parts are sent to remote locations for evaluation, and the results are compiled at the local site. There the results are translated, if necessary, from the universal data model to the local data model, and sent to the user. Once the query evaluation is complete, the ECNDD is updated with the locations of the accessed data that were not currently on the ECNDD.

Servicing Local Updates

Local updates are updates that originate at the local site. The first function in evaluating local updates is to determine the update type. There are two types of updates: updates to host data that exist at the local site that is not replicated elsewhere (unique host updates), and other updates that involve network data (network updates). Each of the local update types has separate activities.

Executing unique host updates includes first, if

necessary, translating the update from the universal database language into the local database language. Next the update is sent to the host computer for processing. The host computer executes the update using its DBMS, and sends the results back to the DDBMS software. Finally, the results are shown to the originating user.

Executing network updates first may require translating the update from the local database language into the universal database language. Then the data locations are searched for on the ECNDD and, if not found, the CNDD site is sent messages requesting the locations. If all the data is at a single site (a unique network update), the update is sent to the site and the update is transacted in a method similar to updating unique host data. If the data is replicated among multiple sites (a replicated network update), then a method must be chosen to keep each of the sites concurrent with one another, and yet keep from causing a deadlock of the DDBMS. The replicated network update case is not decomposed since the design is algorithm-dependent. Once the updates have completed the transactions, any pending updates (updates to copies of data at sites that are currently inactive in the DDBMS) are sent to the CNDD site to be added to those sites' pending update files.

Servicing Remote Queries

Remote queries are queries sent from another site to the site where this routine is executing. Executing remote

queries includes, if necessary, translating the remote queries from the universal database language into the local database language. Next, the translated remote queries are sent to the host computer for evaluation. After the host computer completes the evaluation, the results are returned to the DDBMS software. The results are then translated, if necessary, from the local data model to the universal data model. Finally, the results are sent back to the originating site.

Servicing Remote Updates

Remote updates are updates sent from another site to the site where this routine is executing. Executing remote updates includes, first, examining a flag on the remote update to determine if the update is a unique network update or a replicated network update. Unique network updates are translated, if necessary, from the universal database language into the local database language, and sent to the host computer for evaluation. The results are received from the host, and sent back to the requesting site. Replicated network updates are updated using some kind of concurrency-maintaining and deadlock-preventing algorithm, and due to its dependency on a specific algorithm, is not decomposed further in the requirements analysis.

Servicing CNDD Site Requests

CNDD site requests are those messages sent to the site

that contains the CNDD and the pending update files for inactive sites. The routine first determines the type of CNDD site request, and calls the appropriate routine. CNDD data location requests are executed by collecting the CNDD data required by a site, and sending that data to the site. CNDD updates are executed by matching the update against the CNDD data, updating the CNDD, and sending an update acknowledgement back to the sending site. Pending update requests are matched with the pending update file for the site that is currently inactive in the DDBMS, and the request is added to that site's pending update file.

Summary

The Requirements Analysis for a DDBMS system is given using the SADT decomposition technique. Overall modules include initializing the DDBMS at startup time, reconfiguring the DDBMS during execution, and executing the DDBMS software at the individual sites. Various algorithm-specific functions, including the query optimization and the concurrency control functions, were not decomposed to maintain a general character to the requirements analysis. This requirements analysis provides the basis for the detailed design of the AFIT DEL DDBMS.

III. DETAILED DESIGN

Introduction

This chapter presents a detailed design of a DDBMS based on the requirements analysis of the previous chapter. The first task of the design was to make implementation decisions on the requirements to limit the scope of the design in this thesis. To accomplish this, three additional SADT diagrams were generated, along with the associated data dictionary and indexes. This information is in the beginning of Volume III of this thesis, the detailed design document.

Following the completion of all the requirements analysis, a detailed design was made using structure charts and associated process and parameter data dictionary entries. This design is located after the SADT diagrams in Volume III. The structure charts show the basic structure of the software modules to be implemented. The data dictionary entries show the details of implementation with the process entries giving the algorithms, descriptions, and inputs and outputs of modules, and the parameter entries giving the descriptions and data characteristics of data items passed between modules.

The idea of the design was to be as general as possible, but to limit the design in the specified areas according to the implementation decisions made. The generality comes from the fact that a large proportion of

the design is applicable in most DDBMS configurations, and the implementation decisions were made to limit the amount of software design and code written for the query optimization and concurrency control of updates.

Further Decomposition of Requirements

There were three areas where deliberate limitations were placed on the requirements of the DDBMS. Those areas are:

(1) Optimizing network queries

(2) Handling replicated network updates at the originating site

(3) Handling replicated network updates at the remote sites

There are several complicated DDBMS query optimizing schemes available, such as Wong (14:50-68), but to make the design simple enough to be implemented in a short amount of time, shortcuts were necessary. Instead of determining the optimum locations for executing queries, the algorithm merely optimizes the relational structure of the query, gets the locations of the data items needed to solve the query, and partitions the query by sending a query part to the first availible location and receiving perhaps a very large relation for a result.

The problem of controlling the concurrency of updates in a DDBMS is also very complicated, as Thomas shows (12:88-

94). To simplify the process, instead of using a complicated voting scheme that was described by Thomas, the locations of the data in the update are determined and the update is sent to each of the sites. Results are received from those sites as they come in, waiting until a specified timeout period is over. Then the ECNDD is updated with the locations of the data in the update.

At the remote site that receives a replicated network update, instead of voting and waiting for results, the update is translated, if necessary, from the universal database language into the local database language, sent to the host computer for the transaction and the results of the update received from the host computer are sent back to the originating site.

DDBMS Structure Chart Design

Beginning here and continuing for the rest of the chapter is a text description of the structure chart design and the accompanying data dictionary descriptions that are in Volume III. Copies of the software from this design will reside at each site in the DDBMS with the CNDD site software shut off at all sites except the CNDD site. The design is set up so that it could be implemented on any computer, but includes the design restrictions stated in the further decomposition of the DDBMS requirements earlier in this chapter.

The highest level diagram of the structure charts is

shown in Figure 8 under the title "Execute DDBMS at Site N". In this design, Site N is referred to as the site where that particular software module is executing. There are three main functions for this module: initialization of the DDBMS at this site under "Initialize DDBMS at Site N", retrieving the next input message through "Get Next Message", and creating a process for that message and placing it in the process queue under "Create and Queue Process". Each of these will be described in the following paragraphs.

Initialize DDBMS at Site N

Site N can be chosen as either the central site or as another remote site in the DDBMS. If chosen as the central site, first the database data and the CNDD data at Site N are prepared for interaction with the DDBMS. Next, input and output queues are set up for incoming and outgoing messages that deal with both the host computer and the network. Once the queues are established, each of the sites expected to be in the DDBMS entered by the user at startup time are sent a query message making sure they are active. The return messages are retrieved through "Get Next Message", to be discussed in the following section, and the status information table is updated concerning the status of all DDBMS sites. Then, each active site is given a command enabling it to begin DDBMS execution.

If Site N is chosen as a remote, or non-CNDD, site, a

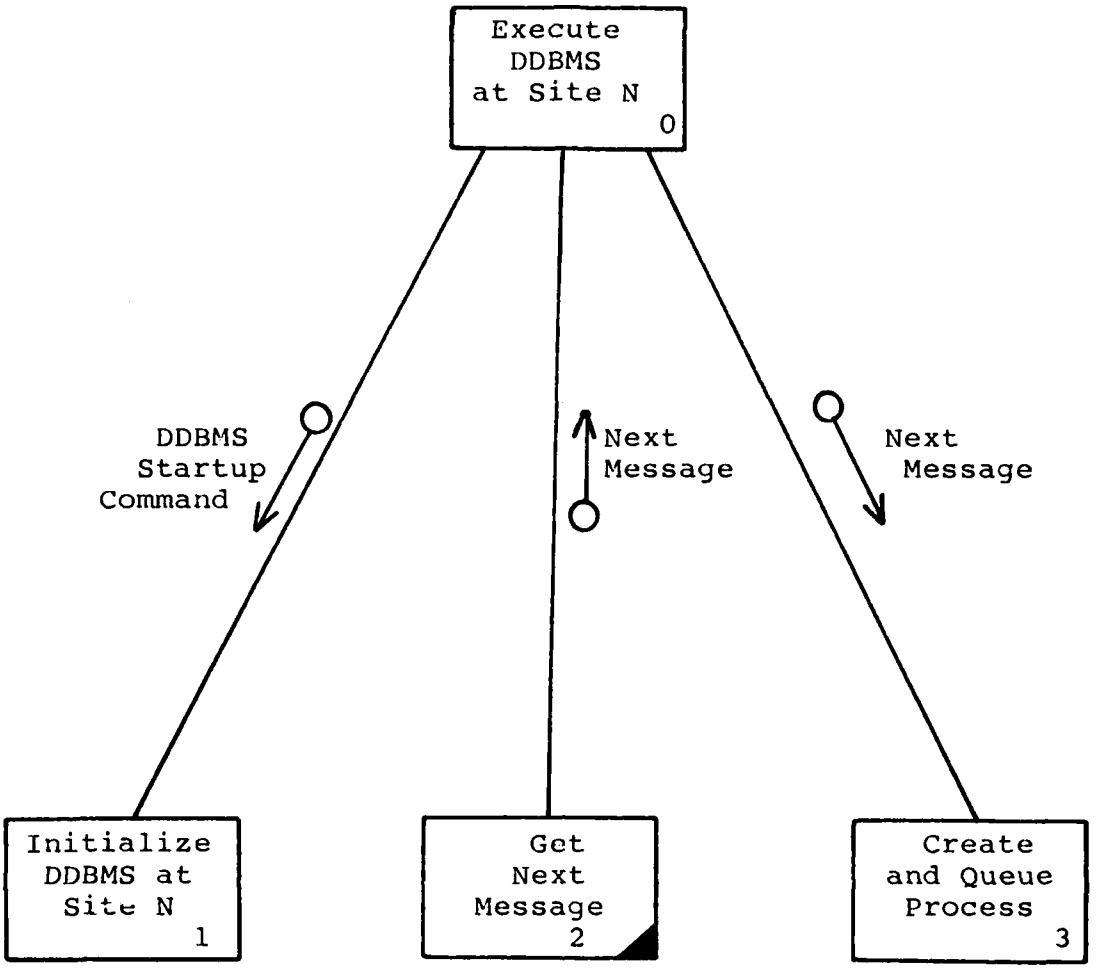


Figure 8 Structure Chart for Process
"Execute DDBMS at Site N"

slightly different sequence of events occurs. First, the database data at Site N is prepared for execution. Next, the input and output queues are prepared in the same manner as in the central site case. The query message to Site N from the central site is retrieved through "Get Next Message", and is used to update Site N's status information

table concerning the location of the central site. Finally, "Get Next Message" is called again to retrieve a command from the central site indicating that the DDBMS is ready for execution.

Get Next Message

The purpose of "Get Next Message" is to retrieve the highest priority input message from the input network and local queues. The routine checks the network and local queues for the message with the highest priority, removes that message from the proper queue, and returns the message to the calling routine. After DDBMS initialization, this message is always sent to "Create and Queue Process".

Create and Queue Process

This routine creates a process for an input message received from "Get Next Message" and places that process in a process queue. The operating system process scheduler determines from the priorities of the processes in the process queue which one to execute next. The execution of the next process is shown in Figure 9 under the title "Execute Process".

The message contained in the executed process is interpreted by "Interpret Network Messages" to determine if the message is a request to reconfigure the DDBMS, or some other type of request made to Site N. If the message is a reconfiguration request then the routine passes the message

to "Service Reconfiguration Requests". Otherwise, the message is passed to "Service Network Messages and Local Requests". Both of these routines are explained in the following sections.

Service Reconfiguration Requests

There are three kinds of reconfiguration requests possible, local reconfiguration requests (those originating at Site N), remote reconfiguration requests (those originating at a site other than Site N when Site N is not the CNDD site), and CNDD site reconfiguration requests (those from another site to Site N when Site N is the CNDD site). There are also four possible reconfigurations to the DDBMS: adding a site, deleting a site, relocating the CNDD site data from one site to another, and recovering from a network malfunction. First, the request type (local, remote, CNDD site) is determined, then the function of the message is determined, and, finally, the function is carried out.

Servicing Local Reconfiguration Requests

Local reconfiguration requests come from the host and are designed to add Site N to the DDBMS, delete Site N from the DDBMS, relocate the CNDD site data from another site to Site N, and to rebuild the CNDD at Site N because of the failure of the CNDD site.

To add Site N to the DDBMS, first Site N sends a

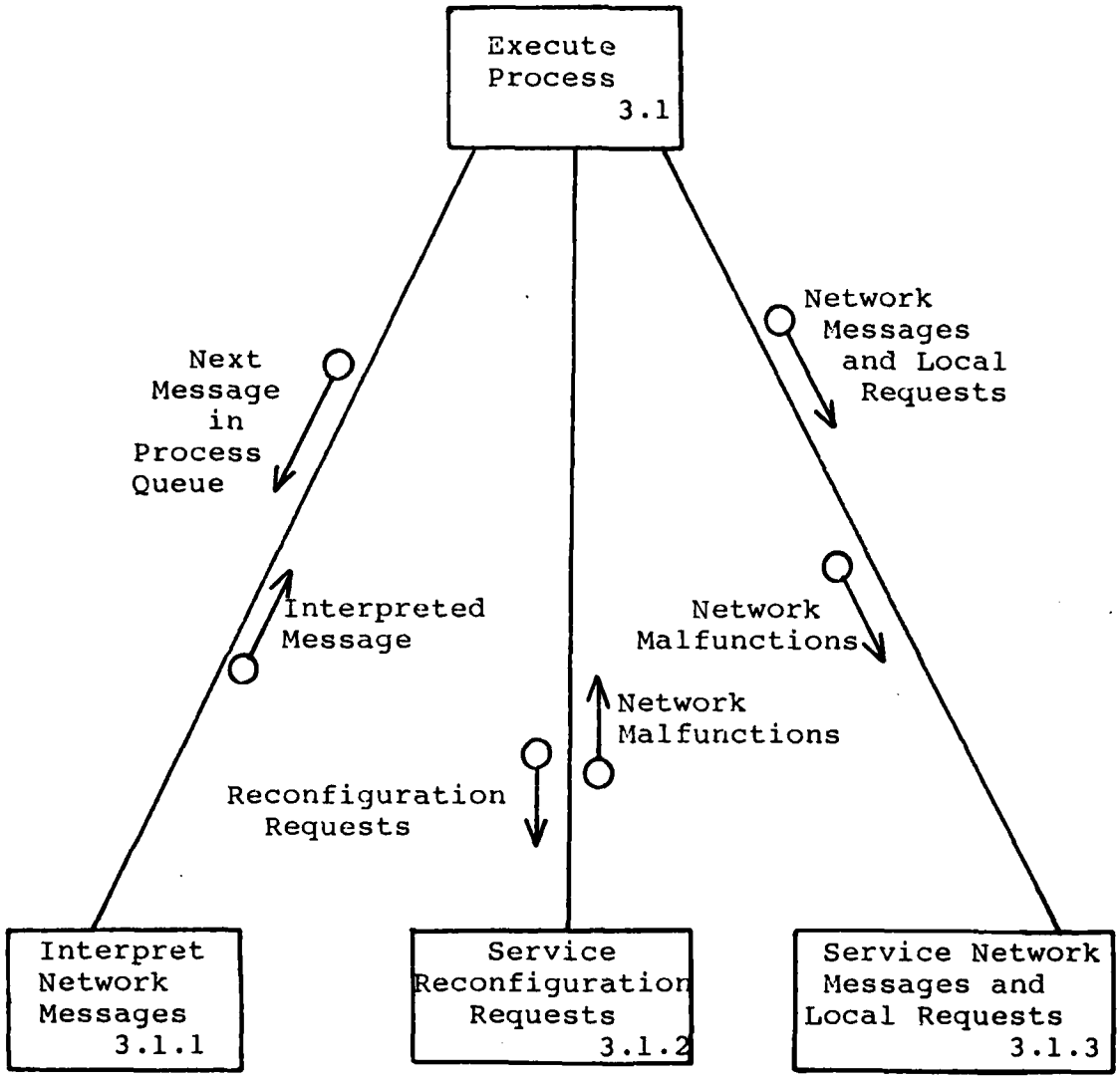


Figure 9 Structure Chart for Process "Execute Process"

message to the CNDD site, whose location is input by the user, indicating Site N's addition. Next, the CNDD site sends a return message, acknowledging Site N's addition.

The return message contains information on the status of other DDBMS sites, which Site N uses to update its status information table. Finally, Site N receives a pending update file from the CNDD site containing updates to data at Site N replicated elsewhere in the DDBMS that occurred while Site N was inactive.

Deleting Site N is much simpler, and only requires two steps. First, Site N's status information table is updated with its own deletion from the DDBMS. Following that, a message is sent to all active sites to mark Site N inactive in their status information tables, and to the CNDD site to begin a pending update file for Site N.

To relocate the CNDD site data from some other site to Site N requires first that a storage area be set aside for that data. When that is complete, Site N sends a message to the old CNDD site indicating that Site N is ready to receive the CNDD site data. When the data is sent, Site N copies the data into the reserved data area. When the copying process is complete, Site N sends a message back to the old CNDD site stating that the process is complete. The status information table is updated at Site N indicating that Site N is now the CNDD site, and Site N sends a message to all active sites to update their status table likewise.

Rebuilding the CNDD at Site N after a CNDD site crash involves constructing a new CNDD from LNDDs at active DDBMS, and updating status tables in the DDBMS to reflect the

change. First, messages are sent to all active DDBMS sites to send their LNDDs to Site N. When they are all received, Site N builds the CNDD from these LNDDs. The status table at Site N is updated to reflect that Site N is now the CNDD site, and a message is sent to all active DDBMS sites to update the status tables as well. Unlike the other local reconfiguration requests, this request can be made from another network site, as well as Site N.

Servicing Remote Reconfiguration Requests

Except for a request to send LNDD data to a site where the CNDD is being rebuilt, all remote reconfiguration requests are used just to update Site N's status information table about events at other sites in the DDBMS. For an LNDD request message, Site N's LNDD data is copied to a temporary file, formatted for transfer, and sent to the site where the CNDD is being rebuilt.

Servicing CNDD Site Reconfiguration Requests

These procedures are executed only if Site N is the CNDD site. There are four possibilities of the CNDD site's role in the reconfiguration of the DDBMS: processing a site addition message, processing a site deletion message, relocating the CNDD data from Site N to another site, and recovering from a malfunction in the DDBMS other than a CNDD site failure.

To process a site addition message, first a return

message is sent from Site N back to the added site. Next, Site N updates its status information table indicating that the site is active. Finally, the pending update file for the site is sent from Site N to that site.

In evaluating a site deletion message, Site N initializes a pending update file for the site about to become inactive. It also updates Site N's status information table indicating that the deleted site is inactive.

The process of relocating the CNDD data from Site N to another site involves receiving a message from the new site indicating it has a place prepared for the CNDD data and pending update files once the message is received, and updating Site N's status information table indicates that the new site is now the CNDD site when a message is retrieving stating that the CNDD copying process is complete.

The last possible task for a CNDD site reconfiguration command is one to recover from a network malfunction that does not involve a CNDD site failure. If a site crashes in the DDBMS, a pending update file is started for that site at Site N. If some network lines are down, this routine establishes communication paths around those lines. Any recovery activity is noted in the Site N status information table and sent to all active sites by a message.

Service Network Messages and Local Requests

The routine to handle all incoming messages except reconfiguration requests is "Service Network Messages and Local Requests", and is decomposed as shown in Figure 10. The overall function of the routine is to determine the type of the incoming message and send the message to "Update ECNDD from CNDD Updates", "Update and Maintain LNDD", or "Service Requests". Then any network malfunctions are processed by either "Decipher Non-CNDD Failures" or "Transmit CNDD Failure Message".

Some input messages are updates to data on the ECNDD at Site N that was just updated at the CNDD. The ECNDD needs to be kept current to prevent it from giving out faulty data locations. In "Update ECNDD from CNDD Updates", these updates are transacted and acknowledgements are sent back to the CNDD site acknowledging the transaction.

Other input messages are LNDD updates, either input externally from a user or input from the CNDD site where replicated data at Site N's LNDD was updated. In either case the update is made to the LNDD and results are sent to the host computer at Site N to notify the user and keep the local database format current. For LNDD updates that originate at Site N, updates are sent to the CNDD site and acknowledgements are received from the CNDD site after the CNDD is updated.

All other input messages are sent to "Service

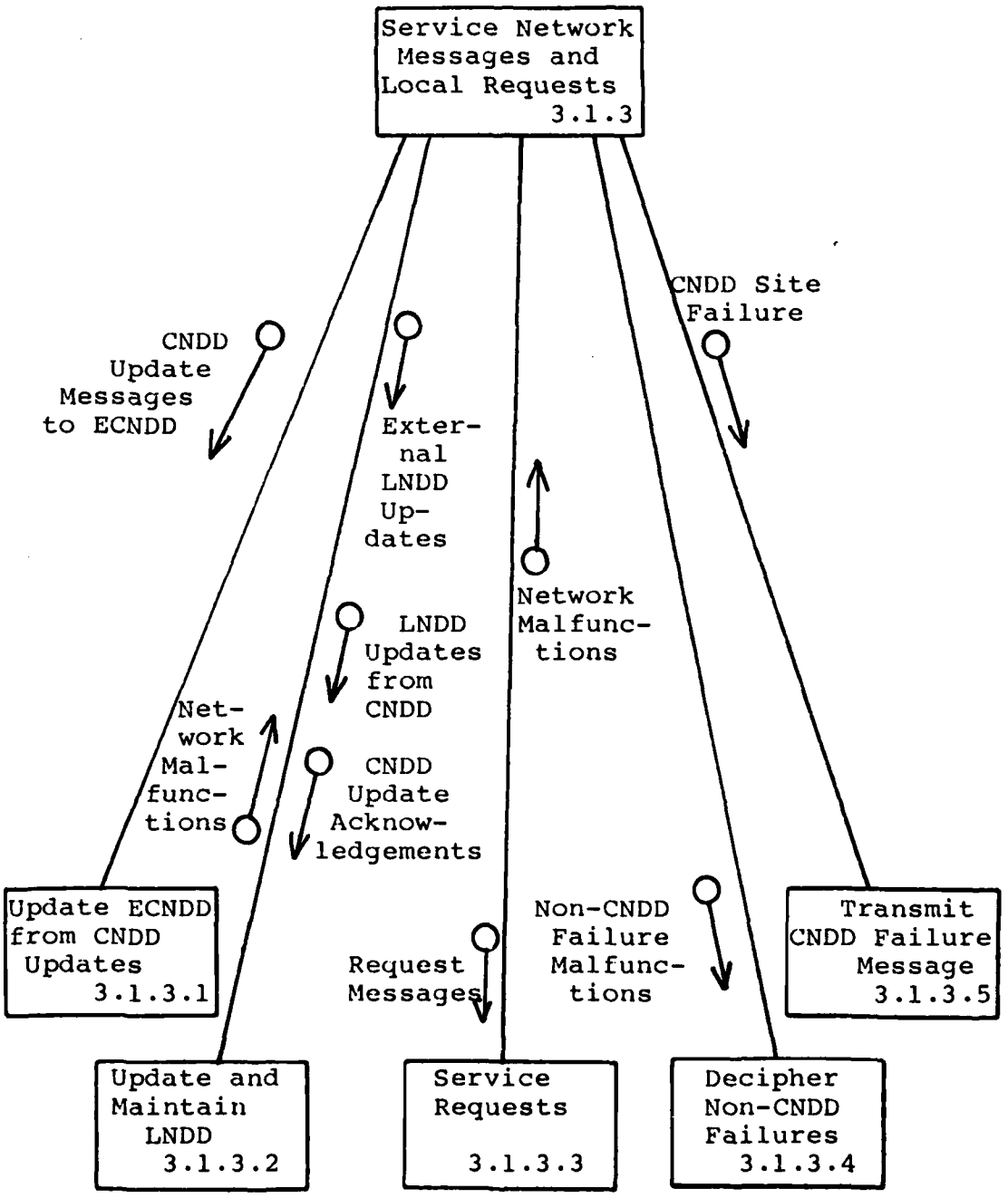


Figure 10 Structure Chart for Process "Service Network Messages and Local Requests"

Requests", which handles queries, updates, associated return messages, and CNDD site requests. The execution of this

routine is explained in the following section.

Network Malfunctions can come from "Service Reconfiguration Requests", "Update and Maintain LNDD", and "Service Request" routines. There are two types of network malfunctions: those in which the CNDD site crashes and those in which it does not. Non-CNDD failure messages are interpreted by "Decipher Non-CNDD Failures" for the actual problems, and a message is sent to the CNDD site with the problem. CNDD site failure messages are sent by "Transmit CNDD Failure Message" to the host computer at Site N and to the site that is next in line to be the CNDD site.

Service Requests

The routine "Service Requests" is decomposed as shown in Figure 11. It sends locally-originating updates and queries and results of the updates to "Service Local Requests". Remote updates and queries and results of the updates from the host computer are sent to "Service Remote Requests". Finally, all requests to the CNDD site when Site N is the CNDD site are sent to "Service CNDD Site Requests".

In "Service Local Requests", there are routines for both local queries and local updates. Likewise in "Service Remote Requests" there are routines for both remote queries and remote updates. And in "Service CNDD Site Requests" there are routines to handle requests to the CNDD for data locations, updates to the CNDD, and requests to add a

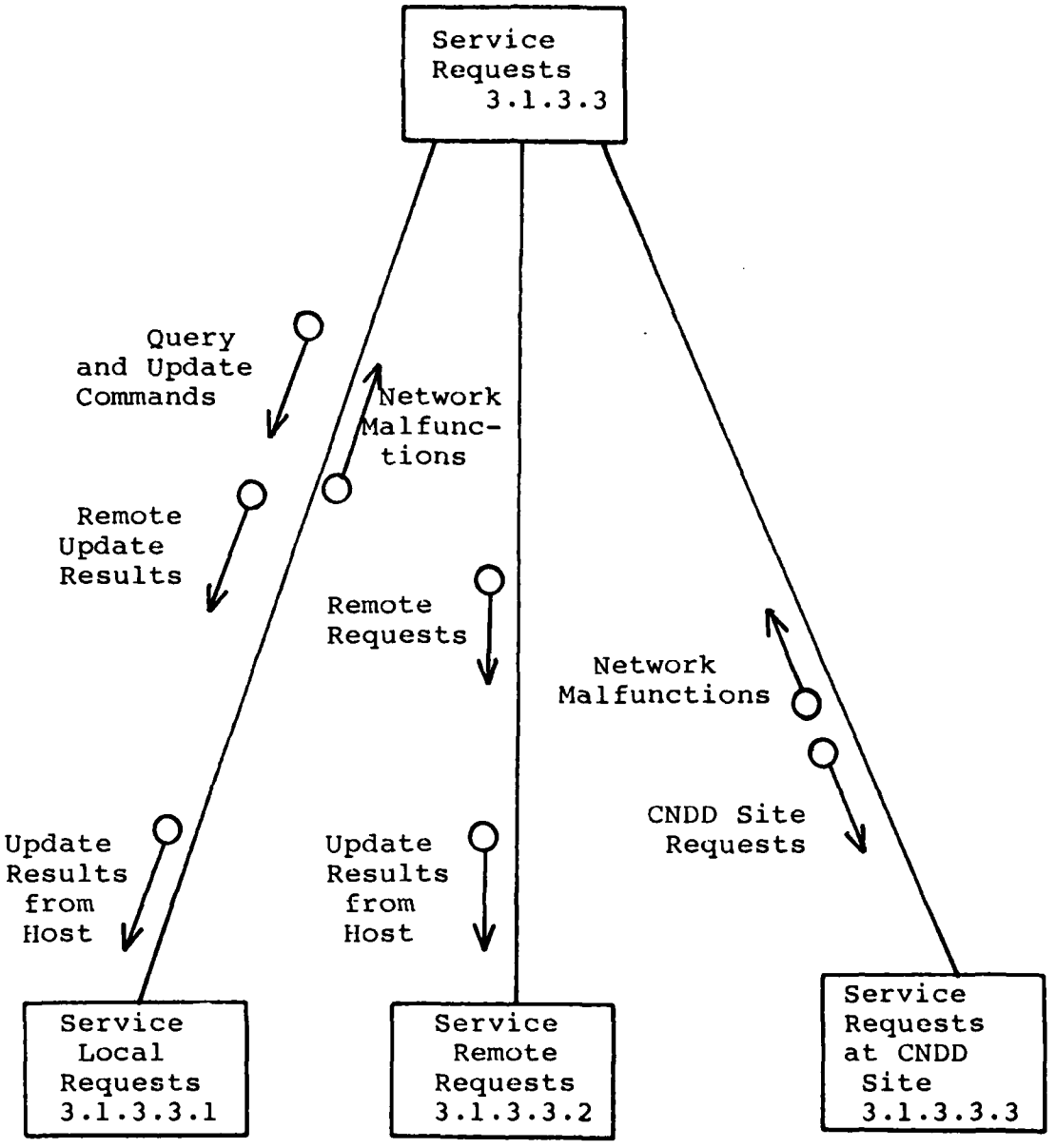


Figure 11 Structure Chart for Process "Service Requests"

pending update to a pending update file for an inactive site. All of these will be explained in the following paragraphs.

Servicing Local Queries

The two types of local queries are those that require only data from the host computer (host queries) and those requiring data from other DDBMS sites (network queries). Host queries are first translated, if necessary, from the universal database language into the local database language. Then they are packaged into messages and sent to the host computer for evaluation. After completion, the results of the query return from the host are translated, if necessary, from the local data model into the universal data model, and are sent as a file to the host computer.

Network queries are translated, if necessary, from the local database language into the universal database language. Locations of the queries are derived from the ECNDD and LNDD, if possible, or sent for and received from the CNDD. Next, the queries are optimized for their structure and decomposed according to locations of required data by a partitioning algorithm. The query parts are sent to the remote locations and the results are stored into files as they arrive. The query result files are combined according to the original query into one result file. The file is translated, if necessary, from the universal data model into the local data model, and sent to the host

computer. Also, the ECNDD is updated with the data locations derived from the CNDD.

Servicing Local Updates

The two general types of local updates are those that update only data at the host computer (unique host updates), and those that update data at sites other than the host computer (network updates). Unique host updates are translated, if necessary, from the universal database language into the local database language. Then the update is sent in a message to the host computer. A return message provides the results of the update which are sent back to the host computer.

Network updates are translated, if necessary, from the local database language into the universal database language. Next, the update locations are derived from the ECNDD, LNDD, and, if needed, the CNDD. If the transaction updates unique data, then it is packaged into a message and sent to the location, the results are received from that location, and the ECNDD is updated if the data location was derived from the CNDD. If the transaction updates replicated data, then the updated data is sent to all sites where it is replicated, the results are returned from those sites, and pending updates are generated for those sites which are inactive and are sent to the CNDD site. Also the ECNDD is updated with the CNDD-derived data locations. The

results, either unique or replicated, are lastly sent to the host computer.

Servicing Remote Queries

Remote queries are translated, if necessary, from the universal database language into the local database language. Then they are packaged into messages and sent to the host computer. The query results from the host computer are translated from the local data model into the universal data model and sent back to the originating site.

Servicing Remote Updates

There are two types of remote updates: those whose data is unique at Site N, and those whose data is replicated elsewhere in the DDBMS. In this implementation they are both executed in the same manner. The updates are translated, if necessary, from the universal database language into the local database language. Next, the updates are packaged into messages and sent to the host computer. Then the results from the host are sent in a message back to the originating site.

Service CNDD Site Requests

There are three types of CNDD site requests: CNDD data location requests, CNDD updates, and pending update requests. When a data location request is received, first the CNDD is scanned and the required data locations are retrieved. Then the data locations are formatted and sent

as a file back to the originating site.

Updates to the CNDD are from sites whose LNDD was recently updated. First, the updates are run against the CNDD. Next, the updates are sent to ECNDDs and LNDDs with changed data. If the input message is an acknowledgement from a site whose ECNDD has been updated, then a timeout is shut off for that site; otherwise if a timeout does occur, network malfunctions are reported.

Pending update requests are used to add the pending update contained in the request to the appropriate pending update file for the inactive site. Following this, results of the update are sent in a message back to the originating site.

Message Formats

Formats for messages sent between DDBMS sites and between the host and the DDBMS processor are located in Appendix C. They are listed in alphabetical order and contain priorities for their execution order. The general levels of priority are, first, reconfiguration messages, second, the routine data handling messages, and, third, acknowledgement messages. Further decompositions of priorities are discussed in Appendix C.

Summary

A detailed design of a DDBMS is presented in this chapter. Implementation restrictions were made to the

requirements analysis through further SADT diagrams and the requirements set forth were interpreted into structure charts and associated data dictionary entries. With the completed design the implementation of two communicating DDBMS nodes could begin.

IV. PARTIAL IMPLEMENTATION

Introduction

A part of the DDBMS structure chart design discussed in the previous chapter was implemented. Two LSI-11 microcomputers in the LSINET of AFIT were chosen as host interfaces to hold the DDBMS software that would obtain query data from two hosts: an S-100 computer executing the dBASE II relational DBMS and a VAX 11/780 running the UNIX operating system with INGRES, also a relational DBMS. A relational DBMS language known at AFIT as Roth's Relational DBMS (10:110-135) was used for all input queries, playing the role of a universal language. No data updates of any kind can be sent over the network under this implementation. The structure charts and data dictionary that explain which modules were implemented are in Volume IV of this thesis, Implementation Design.

First, this chapter discusses the architecture of the computers configured in this small network to test out the DDBMS software. Next, techniques of translating queries from Roth's Relational DBMS to dBASE II and INGRES are discussed. Following this are module implementation decisions made to limit the number of modules implemented as a part of this thesis to conform to time constraints. The actual implementation of the DDBMS software is then discussed, followed by a summary of activities occurring during the implementation.

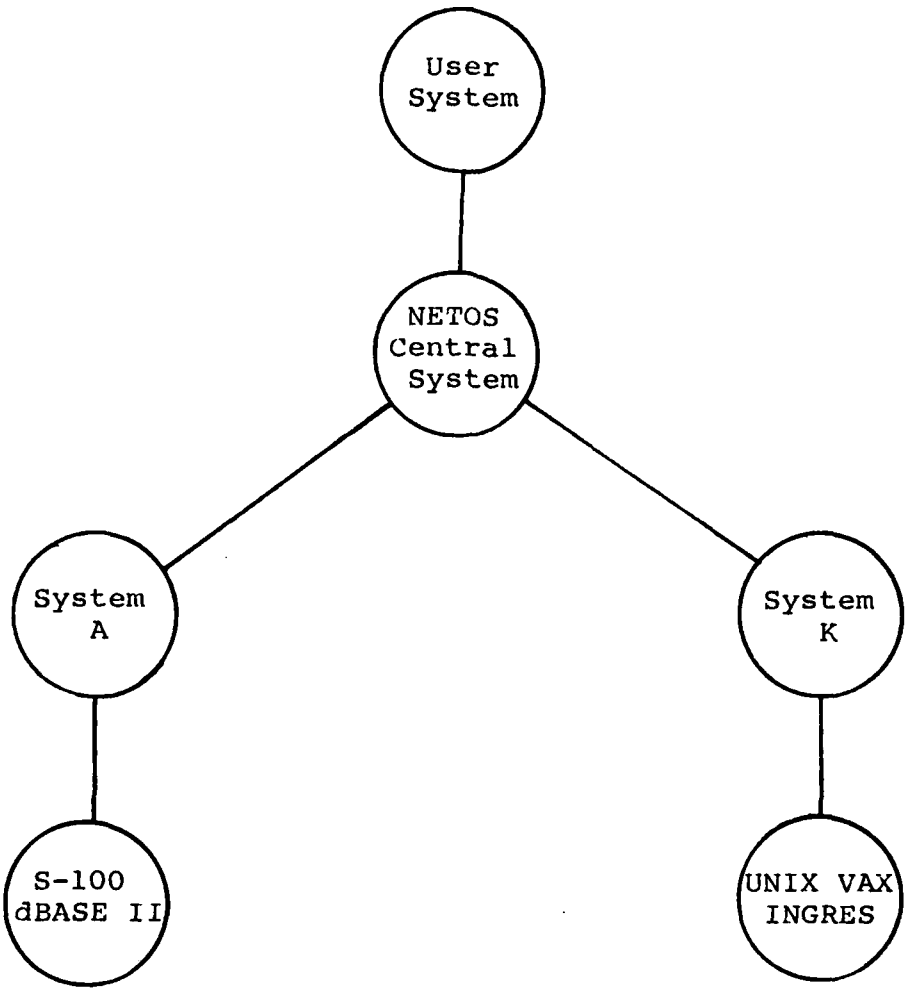


Figure 12 DDBMS Implementation Architecture

Implemented Architecture

Figure 12 shows a pictoral representation of the architecture to implement the DDBMS architecture. The computer architecture chosen for this partial implementation consists of an S-100 microcomputer running dBASE II; a VAX 11/780 running the UNIX Operating System and the relational

DBMS INGRES; two LSI-11 microcomputers (System A and System K) connected in a network via the Network Operating System (NETOS) (4:1-19) where System A is connected to the S-100 and System K is connected to the VAX; a third LSI-11 computer, System B, which acts as the central system in NETOS through which all network messages are passed; and a final "user" LSI-11 computer where queries originate and where the results appear.

Only transfer and execution of queries has been accomplished in this implementation. System A and System K will have identical software except for the commands sent to the respective dBASE II and INGRES systems to translate and execute a query and the software to handle multiple-site queries. The systems will contain algorithms to update and retrieve data only from the ECNDD and the LNDD, and the ECNDD is assumed to have all network location data. All input queries originate at the user system and are formatted in Roth's Relational DBMS language (10:122-124) by a program that prompts the user for relation names, attribute names, and conditions. A small explanation of this language is in Appendix D. Queries are sent to either the S-100 computer with dBASE II, or the VAX computer with INGRES. System A and System K are connected via the central system in NETOS, and send queries to each other to retrieve data from the host computer (the S-100 or the VAX 11/780) of the other site.

The S-100 computer and System A constitute the first DDBMS site. Queries from the user site to System A and those from System K are, in turn, sent to the S-100 computer for computation. At the S-100 computer input queries are translated by a routine written by Gunning (3:1-23) from Roth's Relational DBMS language to dBASE II. The results from dBASE II are returned to System A.

The VAX 11/780 computer with INGRES and System K constitute the second DDBMS site. Queries from the user site to System K and those sent to System K from System A, are sent, in turn, to the VAX with INGRES which translates the query from Roth's Relational DBMS Language to INGRES, executes the query, and returns the results to System K.

Translator Modules

There are two main translation modules involved in this partial implementation of DDBMS software. The first module is one that translates Roth's Relational DBMS queries into dBASE II command files that can execute on the S-100 computer. This module was written by Gunning in December 1983 as a part of an introductory database course at the Air Force Institute of Technology (3:2-3). The second module is a version of the dBASE II translator modified as part of this thesis effort to be used by INGRES under UNIX running on the VAX 11/780 computer. Since the Gunning translator could only handle queries and not updates, it was decided early in the implementation phase to direct the main effort

toward transferring only queries and query results by the DDBMS software.

Figure 13 shows the overall design of both translators: first, to read in a query; second, to break out the relations, attributes, and conditions; and third, to execute the query. In the Roth-dBASE II translator, the Roth Relational DBMS queries are read as input to the program and are segregated by query type (JOIN, PROJECT, SELECT, DIFFERENCE, UNION, INTERSECTION, PRODUCT and DIVIDE commands). Each query type has its individual variables stored into a dBASE II command file, and that dBASE II command file is executed with the results being passed back to the originating site.

The Roth-INGRES translator took the basic structure of the code from the dBASE II translator, but limited the translation capabilities to JOIN, SELECT, and PROJECT queries to simplify operations. Instead of sending the variables to dBASE II command files, the Roth-INGRES translator uses these as input for the QUEL command language (15:EQUEL:1-5) that is imbedded in code written in the "C" programming language in a combination known as EQUEL. The variables are accepted into the EQUEL statements and are executed, with the results returned to System K.

Module Implementation Decisions

Due to the large nature of the DDBMS design and the

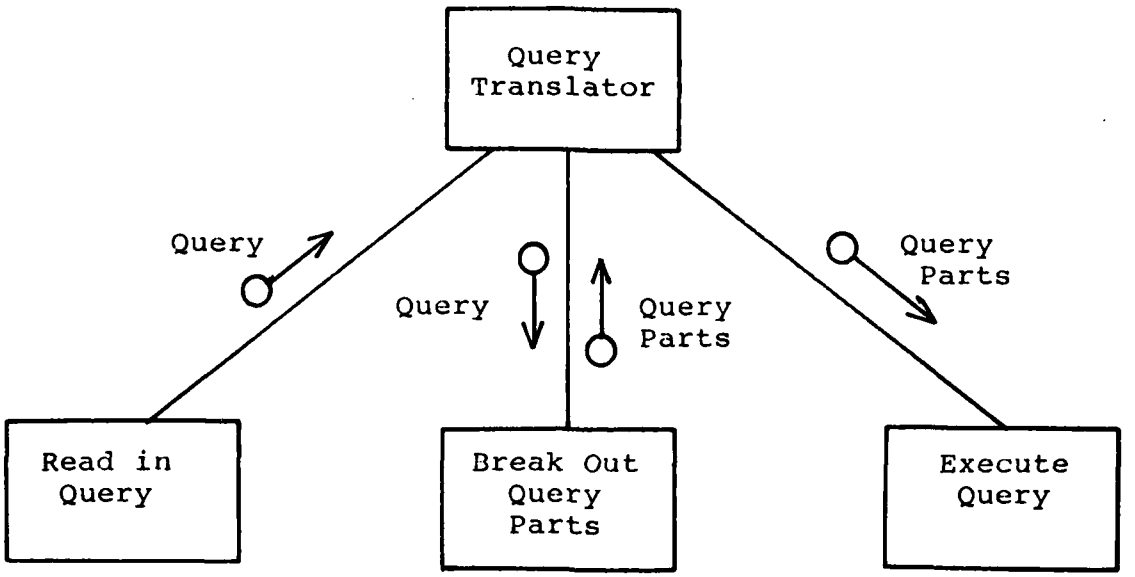


Figure 13 Translator Design

small amount of time with which to accomplish this partial implementation, only a small portion of the actual structure chart design from Chapter 3 was implemented. The main idea was to limit the implementation to only queries and query results being passed over the network in the DDBMS. The LNDD and the ECNDD were implemented as flat files updated by a screen editor. The LNDD contains only data names and associated databases located at the host site, and the ECNDD contains relation names, database names, and network site locations. The ECNDD is assumed to have data on all relations in the DDBMS.

All mechanisms to recover from DDBMS crashes were left

out of this implementation along with all maintenance of pending update files. Also, all data updates were left out of this implementation due to the lack of an effective update translator and the inherent difficulty in keeping updates occurring concurrently. The structure charts and data dictionary for the modules that were coded and tested under this project are in Volume IV of this thesis, and will be discussed in the following sections.

Implementation Design

The implementation design was much smaller that the original design, and some modifications and extensions were made to deal with the constraints of the computers in the implementation architecture. A structure chart of the first-level modules of the DDBMS software is shown in Figure 14. The main executing module calls "Get Next Message", which waits for the next query coming into the system, either from a user site or from another DDBMS site through a call to the "recv_file" module to receive a file over the network. This module is in ISO Layer 6 of NETOS (4:17). The "Get Next Message" module copies the incoming query into a buffer that is passed back to the main executing module and over to "Service Requests".

"Service Requests" handles all incoming query messages. This module retrieves the source from the message, services the query by calling "Service Queries" and obtaining the filename of the result file, and sends the result file back

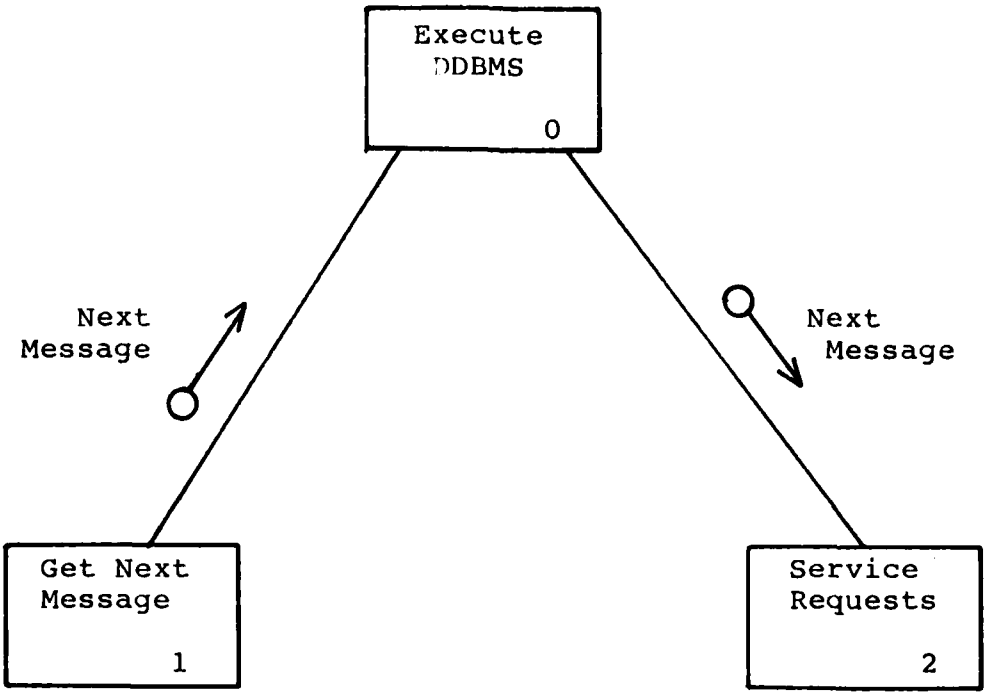


Figure 14 "Execute DDBMS"

to the site indicated by the original source.

The module "Service Queries" is decomposed as in Figure 15. First, the type of query is found in the module "Determine Local Query Type". In this module, relations are extracted from the incoming query and compared with those on the LNDD and ECNDD files, and locations of those relations are returned along with a flag indicating whether the query needs only data from the local site (a host query), or from other sites (a network query). Host queries are handled under "Service Host Queries" and network queries are handled under "Service Network Queries".

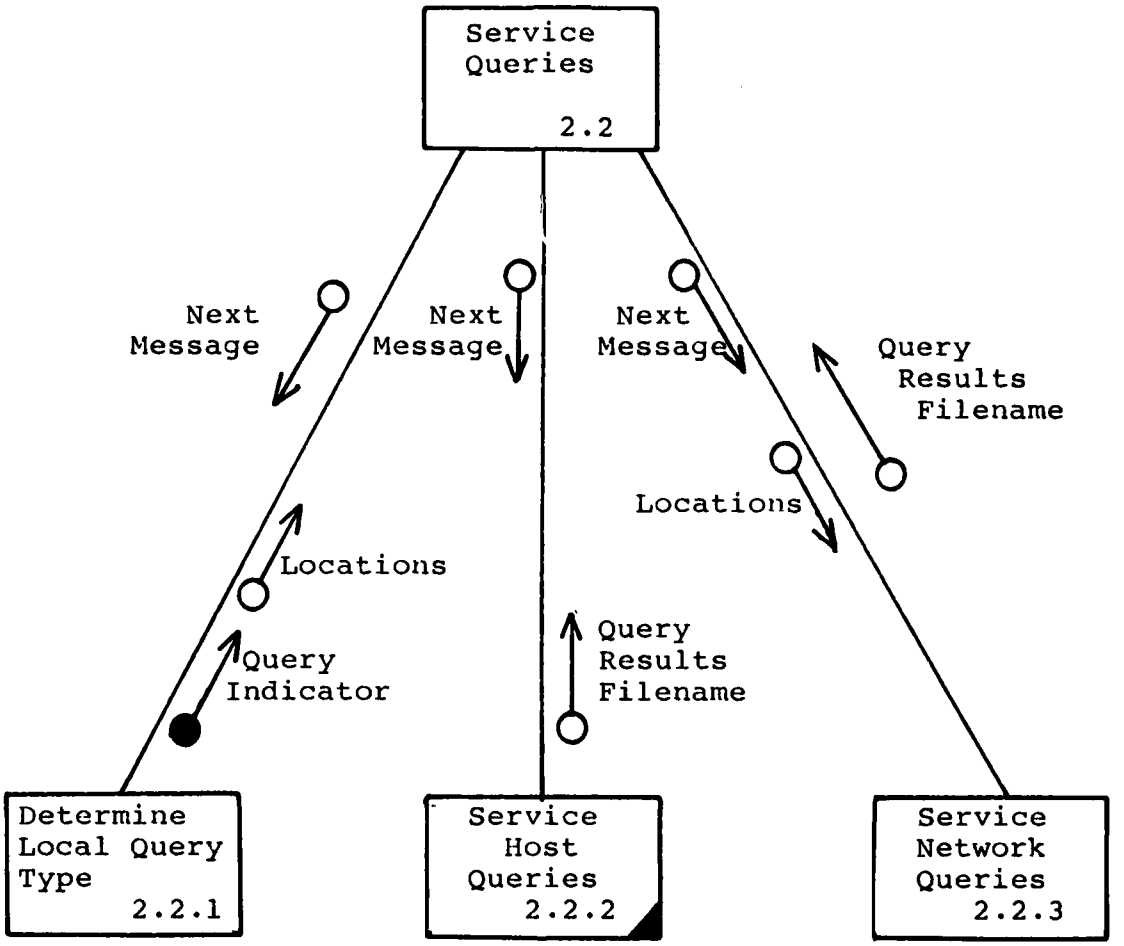


Figure 15 "Service Queries"

Service Host Queries

The module "Service Host Queries" is decomposed into two modules, "Send Host Queries to Host" and "Receive Host Query Results from Host", in Figure 16. The module "Send Host Queries to Host" first gets a filename from a global variable, incrementing the filename in a routine. Then, it writes the query part of the incoming message to the file by

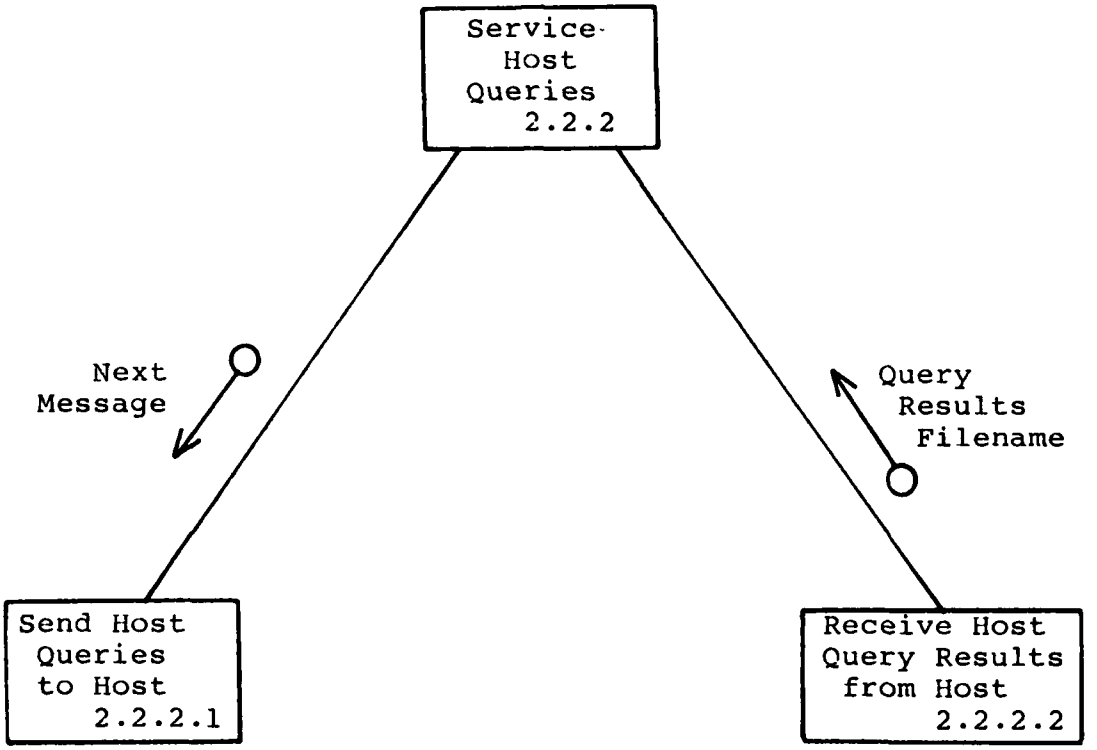


Figure 16 "Service Host Queries"

that filename. Finally, it sends the query file to the host computer using appropriate UNIX or CPM commands. "Receive Host Query Results from Host" gets a filename for the query result file, sends a command to execute the query at the host computer, and uploads the query results from the host computer, again using appropriate UNIX or CPM commands.

Service Network Queries

The module "Service Network Queries" is also decomposed into two modules, "Send Query Parts to Remote Locations" and "Compile Network Query Results", and is shown in Figure 17.

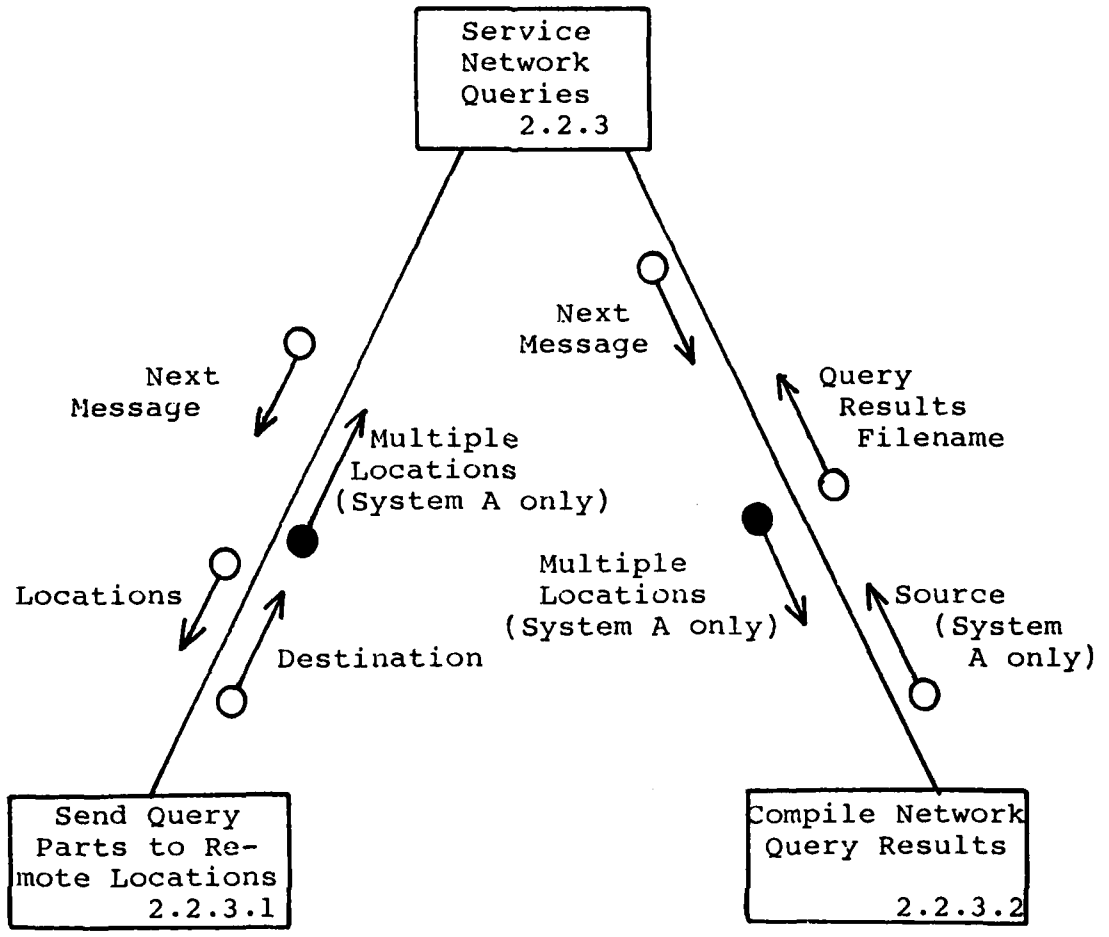


Figure 17 "Service Network Queries"

"Send Query Parts to Remote Locations" gets a filename, builds a remote query message in the file by that filename, and sends that query message to the DDBMS site with the required information.

In this implementation, only JOIN, SELECT and PROJECT commands are possible. SELECT and PROJECT commands need only one input relation, and, since they are network queries, that relation lies completely at another DDBMS site. But a JOIN requires two input relations, and there is

a possibility that these two relations can be spread over two sites. "Send Query Parts to Remote Locations" at System K, connected to the UNIX VAX, sends all multiple-site JOINs to System A, connected to dBASE II on the S-100. "Send Query Parts to Remote Locations" on System A builds a SELECT for the relation that is missing on the S-100 and sends that SELECT to System K. Both routines call the "send_file" function of ISO Layer 6 in NETOS (4:17). Further multiple-site processing is discussed in the next sections.

"Compile Network Query Results" gets a filename and receives the results of a network query into a file by that filename through the "recv_file" function of ISO Layer 6 in NETOS (4:17). The System K version merely returns the resulting filename as the final results. However, the System A version goes through the following process: if the original query was a SELECT, PROJECT, or a single-site JOIN, then the first filename is returned as the final results. But, if the original query is a multiple-site query, then the filename is passed to "Compute Network Query Results" to JOIN together the output from the SELECT with a relation at the S-100.

Compute Network Query Results

"Compute Network Query Results" is implemented only on System A and its decomposition is shown in Figure 18. The

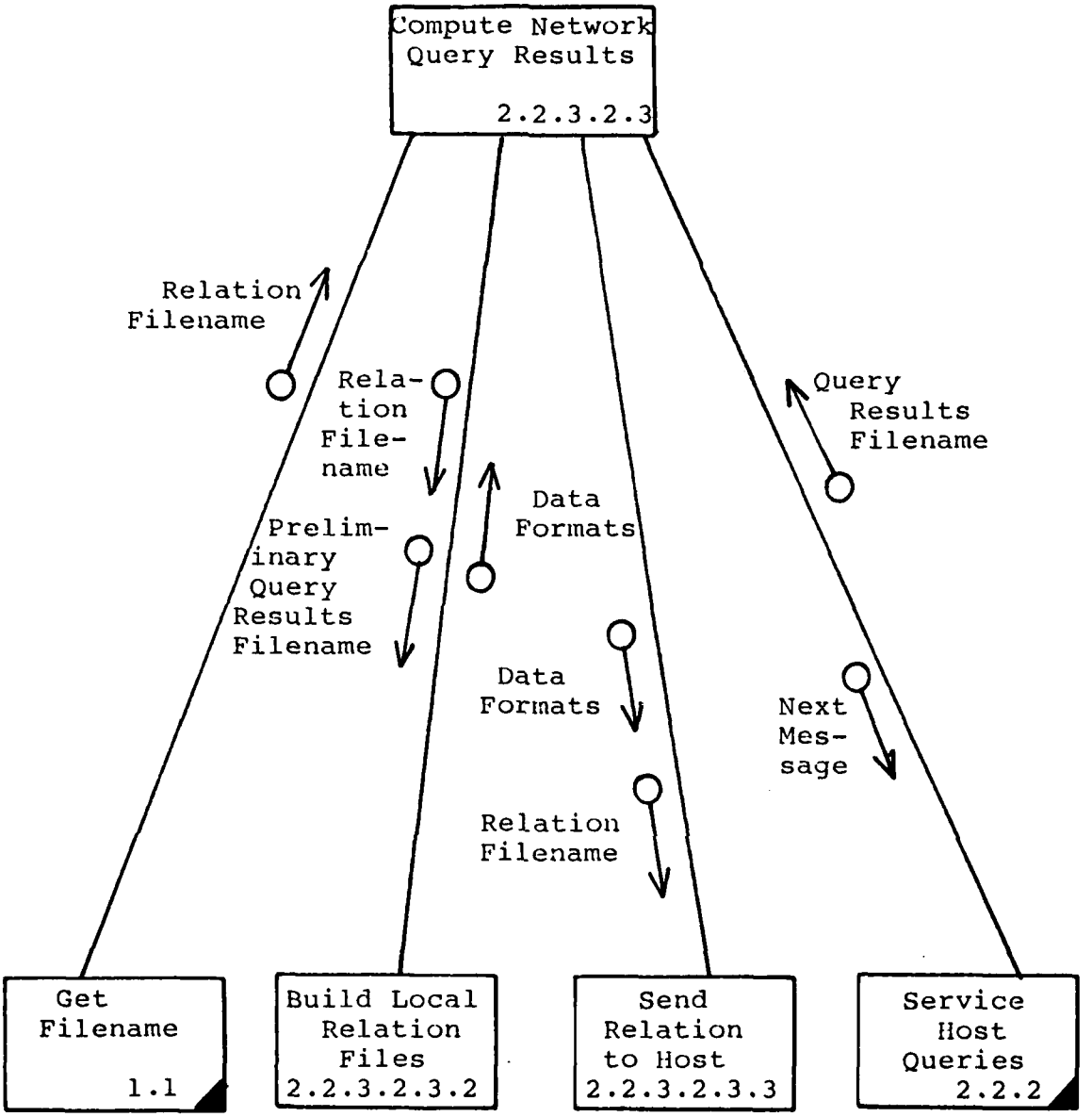


Figure 18 "Compute Network Query Results"

reason that it appears only on System A is that it is much simpler and less time-consuming to create a relation on the S-100 using dBASE II than using the UNIX VAX and INGRES.

First, a filename is obtained using the same "Get

Filename" routine that has been described before. This filename is used to store a resulting relation from the routine "Build Local Relation Files", which converts the SELECT results described previously from INGRES format to dBASE II format. This new file is sent to the S-100 and loaded into dBASE II by "Send Relation to Host". Finally, "Send Host Queries" is executed to process the original JOIN, since both relations are now on the S-100 computer, and the results appear in the file identified by the Query Results Filename.

Summary

A partial DDBMS was implemented in the AFIT Digital Engineering Lab using four LSI-11 computers in the LSINET, an S-100 computer using the CPM Operating System and running dBASE II, and a VAX 11/780 using the UNIX Operating System running INGRES. One LSI-11 computer was a "user" input terminal, where queries were entered and results received, one was the central system for the Network Operating System (NETOS), and the last two (System A and System K) were DDBMS interfaces to the host computers, S-100 and the VAX, respectively. Queries input from the user terminal were in Roth's Relational Database language and were translated by software modules at the host computers into dBASE II and INGRES for execution. DDBMS software was implemented to route queries to locations where the requested data appeared. All modules discussed in this chapter have been

coded and tested successfully, within some constraints, which will be discussed in the following chapter.

V. TESTING AND EVALUATION

Introduction

The software written during the implementation phase was tested to determine its performance accuracy. Cases were devised to test both the quality of the data that was output and the speed of execution. Only the software written for this thesis was tested thoroughly, though modules such as the Roth-dBASE II translator and the NETOS ISO Layers had to perform successfully for the implementation to succeed.

This chapter is divided into three parts. First, test cases for the Roth-INGRES translator will be discussed, as will the test results. Second, test cases for the DDBMS input user site that test all of the DDBMS software modules will be discussed. Third, results of stopwatch tests on the test cases of the second part will be graphed and the results discussed.

Testing of the Roth-INGRES Translator

The modules of the Roth-INGRES translator were tested module-by-module for correctness of output for given input. Then, the overall translator was tested by setting up files with input queries and running these through the translator. The translator breaks out the parts of the query from the Roth input, and executes an equivalent INGRES statement, printing the results on the screen. Figure 19 shows the set

a) JOIN supply, parts WHERE pnum = pnum GIVING newrel

b) JOIN parts, supply WHERE quan > pnum GIVING newrel

c) SELECT ALL FROM parts WHERE ((pnum < 7) and (pnum > 2)) GIVING newrel

d) SELECT ALL FROM parts WHERE ((color = black) or (color = gray)) and weight > 500 GIVING newrel

e) PROJECT supply OVER snum, pnum, jnum GIVING newrel

f) PROJECT parts OVER pnum, pname, color GIVING newrel

Figure 19 List of Roth Statements Used for Testing the Roth-INGRES Translator

of queries that was used to test the translator using the existing relations "parts" and "supply" from the INGRES database "demo". The results of the queries satisfactorily show the accuracy of the translator. It should be emphasized here that the translator can only handle SELECT, PROJECT and JOIN statements, and, in a SELECT statement, the user cannot specify the value of a character field whose elements begin with numbers.

Testing of Overall DDBMS Modules

In the production of the DDBMS software running the LSI-11 computers in the Digital Enginerring Laboratory, each module was again individually tested and connected together for a system test. For this testing, relations were chosen on the S-100 and VAX systems. Figure 20 shows the formats of the LNDDs and the ECNDDs. The relations "parts" and

System A	System K
parts demo	shipment demo
supply demo	bolts demo
	nuts demo

a) LNDD Files

System A	System K
parts demo LSK	parts demo LSK
supply demo LSK	supply demo LSK
shipment demo LSA	shipment demo LSA
bolts demo LSA	bolts demo LSA
nuts demo LSA	nuts demo LSA

b) ECNDD Files

Figure 20 Entries for LNDDs and ECNDDs at System A and System K

"supply" on the VAX, and the relations "shipment", "bolts", and "nuts" on the S-100 were used in the system testing. The LNDD at System K, connected to the VAX, contained entries for "parts" and "supply", and the LNDD at System A, connected to the S-100, contained entries for "shipment", "bolts", and "nuts", as shown in Figure 20a. Both LNDDs contained the database name "demo" for each entry since only one database was used in th s implementation. Both sites had identical ECNDDs, as shown in Figure 20b, containing three-character NETOS location identifiers for the relations and the database used was again "demo" for each relation.

a) JOIN supply, parts WHERE pnum = pnum GIVING newrel
(one site INGRES JOIN)

b) JOIN supply, shipment WHERE snum = snum GIVING newrel
(multiple-site JOIN)

c) JOIN bolts, nuts WHERE type = type GIVING newrel
(one site dBASE II JOIN)

d) SELECT ALL FROM parts WHERE ((color = black) or (color = gray)) and weight > 500 GIVING newrel
(INGRES SELECT)

e) SELECT ALL FROM bolts WHERE (pnum > 30) and (quan > 300) GIVING newrel
(dBASE II SELECT)

f) PROJECT supply OVER snum, pnum, jnum GIVING newrel
(INGRES PROJECT)

g) PROJECT nuts OVER pname, type GIVING newrel
(dBASE II PROJECT)

Figure 21 List of Queries Used in DDBMS Testing

Once the LNDDs, the ECNDDs, and the data relations were properly in place, full scale testing could begin. The list of queries used in the testing is in Figure 21. Note that SELECT, PROJECT, and JOIN queries were chosen so that one of each could be done with data residing at a particular site, and also a multiple-site JOIN is executed. Each of these queries was sent to both System A and System K. A deficiency was noted in translating a SELECT statement using the Roth-dBASE II translator. The translator produced a command file which generated a syntax error for a SELECT condition involving a non-numeric data field. The results of these queries were evaluated to be correct.

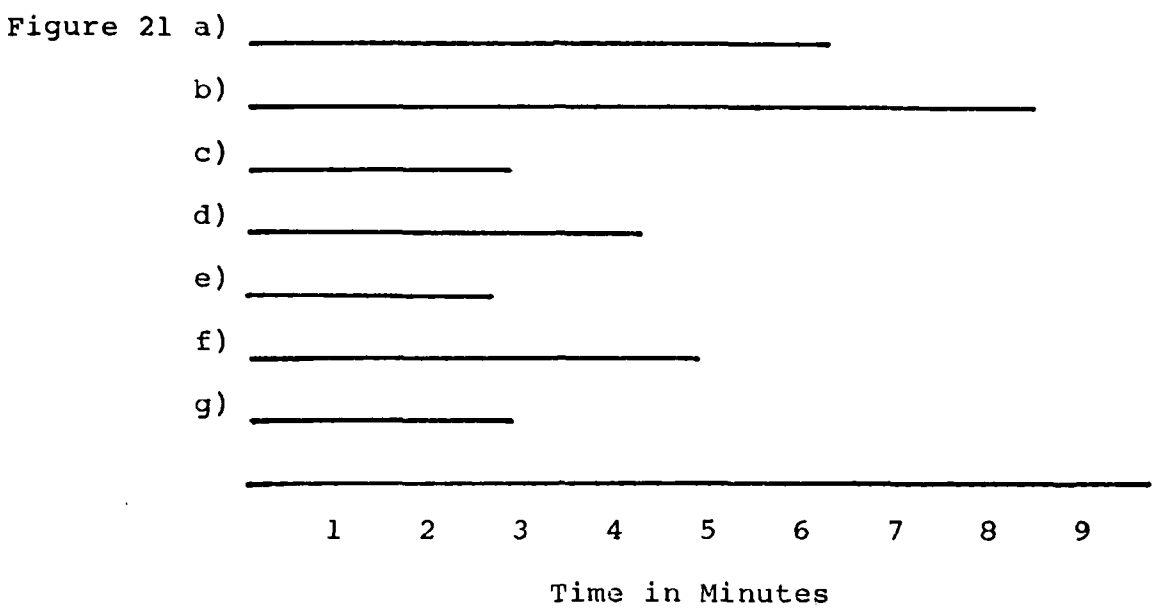


Figure 22 Graph of Total Execution Times for DDBMS Queries

Stopwatch Tests of DDBMS Test Cases

Figure 22 is a graph of the execution times of the queries from Figure 21 run in the DDBMS. All queries in this implementation are sent to System A to simplify processing of multiple-site joins. Note the relative slowness of the queries that required data from the UNIX VAX (frequently overloaded) compared to those that did not require UNIX VAX data. Also note that queries requiring data from more than one computer ran longer than those which required data from only one computer. Finally, note that queries requiring data from the other computer ran longer than those just needing data from that site's host computer.

Summary

The modules involved with the Roth-INGRES translator and the overall DDBMS software were thoroughly tested according to the specifications in this chapter. These queries produced satisfactory results. The queries were timed for their speed of execution and the results were reported. From the design, implementation, and testing of this DDBMS partial implementation, came many ideas for future projects, which will be discussed in the next chapter.

VI. RECOMMENDATIONS AND CONCLUSIONS

Introduction

The previous chapters brought to light many of the problems with this initial design of a DDBMS for use in the AFIT Digital Engineering Laboratory. Many of these problems can be worked out by projects in individual, concentrated areas of focus. This type of focus was impossible while trying to design an entire DDBMS. Each of these projects could become a complete thesis project, part of a thesis, or a class project at AFIT.

The first part of this chapter provides some conclusions about this DDBMS. Important points are brought out about major facts that were learned during the research. Also, major points of DDBMS software will be reemphasized.

The second part of this chapter will introduce possible future DDBMS project topics. First, an update translator is needed in the DEL. Second, the functions of the CNDD and maintaining pending updates need to be implemented. Third, DDBMS update software should be implemented to include an update concurrency algorithm. Fourth, a DDBMS query optimization algorithm is required to direct query parts to optimum sites in the DDBMS. Fifth, software should be implemented to reconfigure the DDBMS when needed and in case of possible catastrophe. Sixth, more sophisticated translators are essential to handling more complicated relational DBMS statements and in the conversion of

statements from a network or hierarchical DBMS language to a relational one. Seventh, algorithms should be implemented to handle input, process, and output queues to transfer messages between DDBMS sites. Finally, once the above projects are finished, the system needs to be converted to be heterogeneous, that is, to handle multiple types of DBMS languages as inputs.

The chapter will end with some final comments on this project. Overall conclusions will be brought forth and major problems will be highlighted.

Conclusions About the DDBMS Research

This thesis effort barely scratched the surface of a huge and widely expanding distributed database research area. It provided a design basis for the projects mentioned earlier in this chapter to continue research at AFIT in this area. The field is very wide and demanding, too large to cover completely in one thesis.

Many major DDBMS areas need to be researched further, and those will be expanded later in this chapter. Also the AFIT Digital Engineering Lab needs more modern network processors than the LSI-11 computers to implement any kind of complicated DDBMS algorithms, preferably multi-programming systems. Research in this area will probably take many years but could, given the proper resources, produce something useful to the distributed database

research field.

Update Translators

One of the things that prevented the implementation of update software in the DDBMS during this project was the lack of an update translator. A suggested type of update translator could parallel the query translators already implemented between the Roth language and INGRES, and the Roth language and dBASE II. The translators could start with the EDIT commands of Roth (10:119-121), and devise appropriate commands in INGRES and dBASE II to accomplish these tasks. Later these translators could be modified into something that could handle updates in a heterogeneous DDBMS environment.

CNDD and Pending Update Software

The structure chart design in Chapter 3 and Volume III of this thesis goes into a great deal of detail about the method of performing CNDD updates, handling CNDD data location requests, and maintaining a file of pending updates for each inactive site in the DDBMS. An overall design for this software could begin with the diagram in Figure 23. A more implementation-specific set of structure charts could be written based on the design done in this thesis. Then, perhaps a node in the NETOS network could be chosen as the CNDD site, and this software could handle requests for the CNDD site as written in the design.

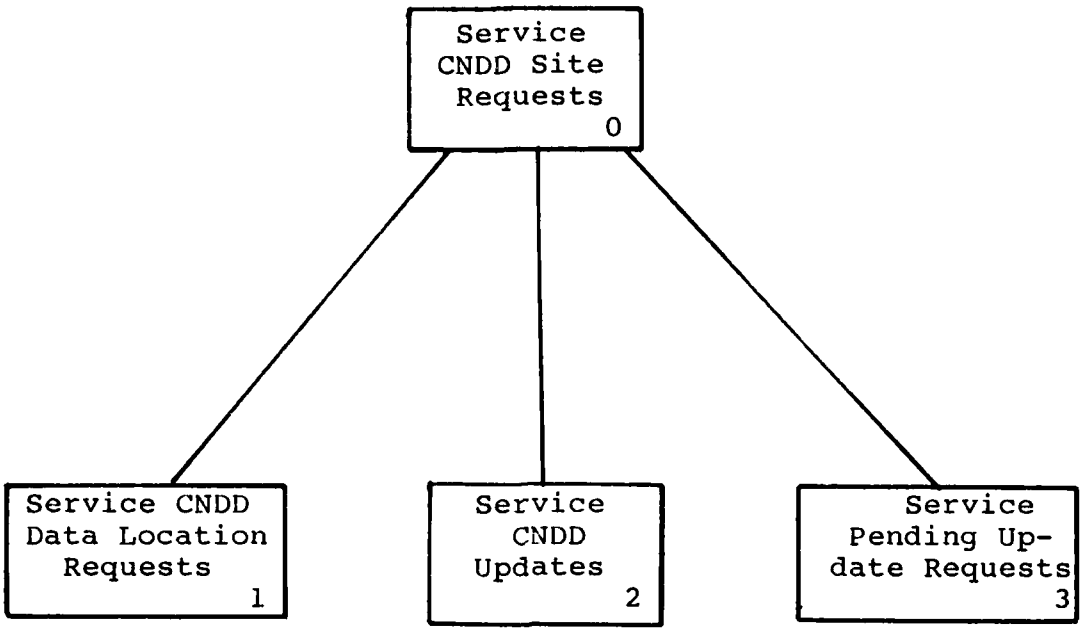


Figure 23 Design of CNDD and Pending Update Software

Update Concurrency Algorithm

Thomas (12:88-94) discusses an update concurrency algorithm that is mentioned in Chapter 1 of this thesis. The design of update DDBMS software in this thesis could be expanded to include such items as timestamping and voting in a concurrency-control algorithm. A small one of these algorithms could be implemented under NETOS in the Digital Engineering Laboratory. This area is very active in the distributed database research field, and perhaps a research could find newer, more advanced algorithms in the literature.

DDBMS Query Optimization Algorithm

To be at all useful, the DDBMS query design of this thesis must be expanded to include optimizing by partitioning input queries, efficient routing of the query parts in the DDBMS, and the final computation of query results. Figure 24 shows the modules that could be found in an optimization scheme.

Partitioning using Roth's language is rather straightforward since no input statement can have more than two input relations and missing relations can be requested by SELECT statements for the whole relation. Any language other than Roth's could require a much more complicated algorithm to implement it.

The routing algorithm for query parts would involve determining which DDBMS site might be available to process a query at any given time, and send a query part to the optimum site. It would also take into account communication bottlenecks and try to ship the query around such bottlenecks. In the NETOS network, this could involve checking ports for availability and sending a query with data at more than one site to a particular site that was free instead of one which was busy.

Computing the final query results from queries in Roth's language involved just loading the results of SELECTs to get needed relations, and executing the original statement. This is very similar to the implementation of

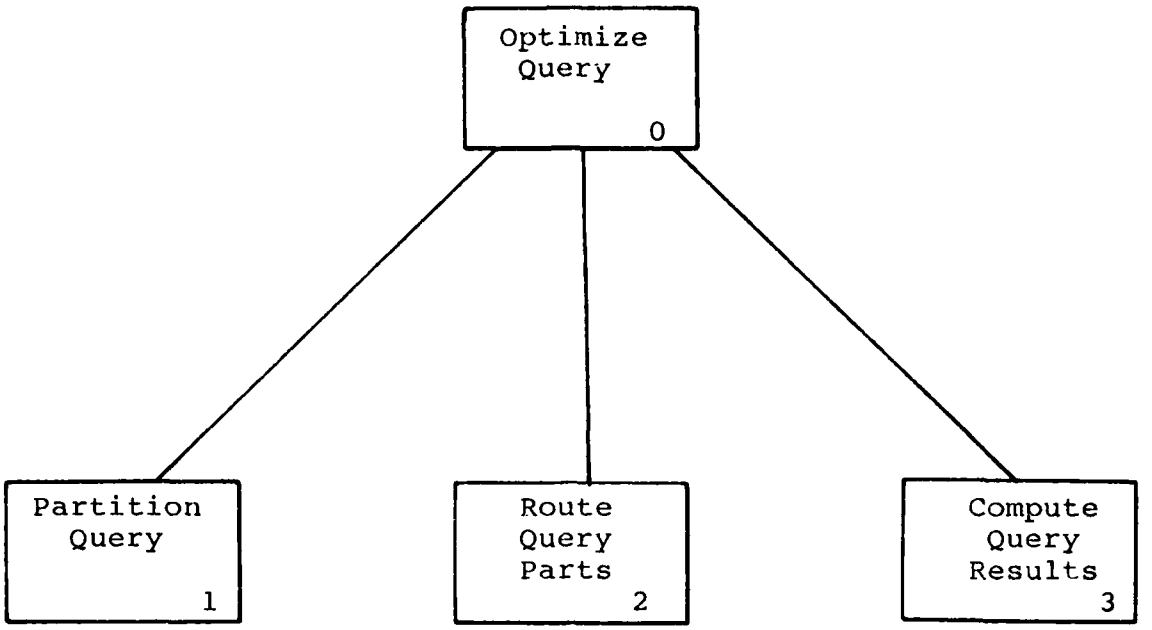


Figure 24 Design of DDBMS Query Optimizer

this function in this thesis project. However, using any language more complicated could be quite a challenge indeed, sending numerous relations and queries around the network until a result was final.

DDBMS Reconfiguration Software

Large sections of Volume II and Volume III of this thesis were devoted to an in-depth design of what would be needed to reconfigure the DDBMS if a site was added, a site was deleted, the CNDD were to be moved to another site, or the CNDD site were to unexpectedly crash. The diagram for this function appears in Figure 25. The design is fairly well intact and, with minimal changes, could be implemented

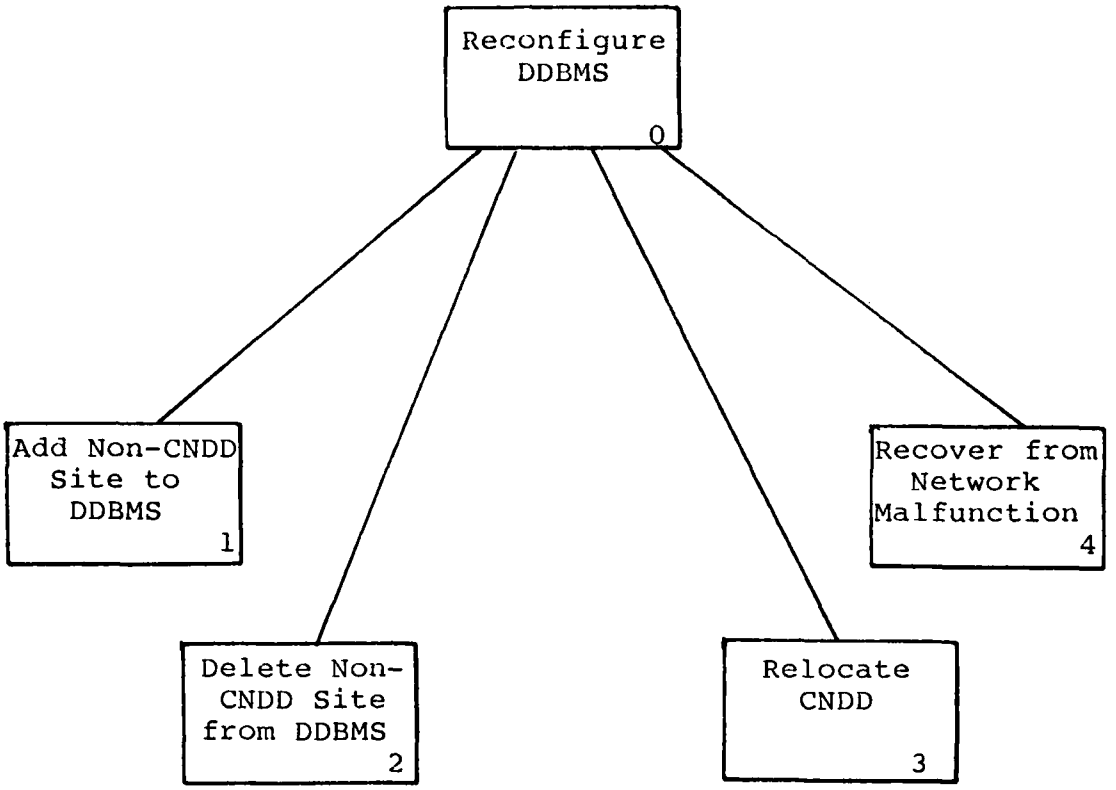


Figure 25 Design for DDBMS Reconfiguration Software

without too much difficulty. In any usable DDBMS, this software is absolutely necessary for day-to-day operations.

More Sophisticated Translators

For any usefulness to come out of the DDBMS software, there must be more usable and efficient translators to different types of DBMSs. The current translators, the Roth-dBASE II and the Roth-INGRES translators, are slow and incomplete. These translators should be completed, and other, more efficient translators should be designed and implemented. Translators could be written from more

difficult languages than Roth's if another language were to be used as a universal language in the DDBMS.

Queue Processing Algorithms

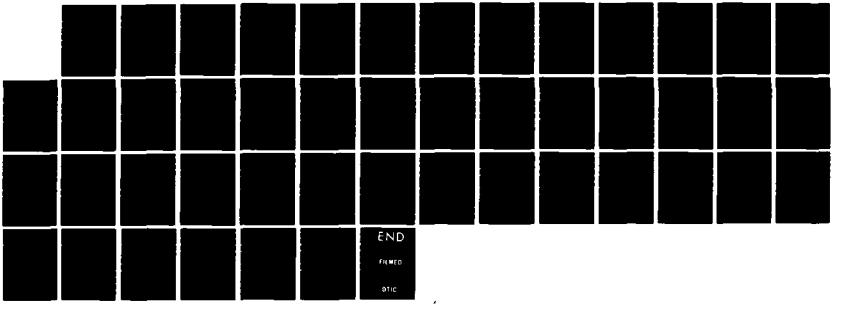
Queues in DDBMS processing could work to store input messages from other DDBMS sites to the one where the software is running, to store processes containing input messages waiting to be started by a system process scheduler, and to store output messages waiting to be sent to another DDBMS site. These queues would allow for a much greater throughput of messages and data in the DDBMS, keeping most sites busy if the influx of requests were great.

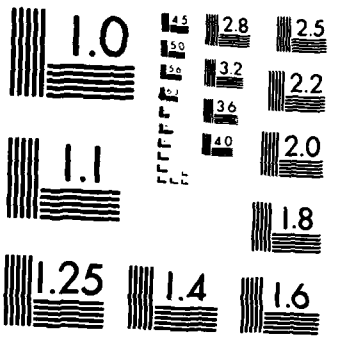
The input queue could line up messages as they come into a system instead of putting the sending site into a wait state while the receiving site finished with the previous message. It would receive a message from another site, copy the message into the queue, then, when the priority of that message becomes the highest in the queue, it would be released. The message would be executed if it were a reconfiguration request, or sent to a process queue if it were some other kind of request.

The process queue holds messages until they are released by a process scheduler to be executed by the DDBMS software. To implement this queue, it is assumed that the DDBMS software is running on a multi-programming computer. The process executes, and any output messages are sent to an

AD-A151 694 DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUT..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J G BOECKMAN 2/2

UNCLASSIFIED DEC 84 AFIT/GCS/ENG/84D-5 F/G 9/2 NL





output queue.

The output queue holds messages until it is their turn to be sent over the network to their destination. Thus, the output software will not be tied up with a single message. The output queue together with the input queue makes the process of passing messages over the network much less time consuming.

Heterogeneous DDBMS Software

The ultimate goal of the AFIT DDBMS project is to create a working, fully operational, heterogeneous DDBMS in the Digital Engineering Laboratory. After most of the homogeneous projects are finished, work can begin converting the homogeneous DDBMS to one that is heterogeneous, where different types of DBMSs will be connected. The areas that will be large problems are translators from network and hierarchical languages into a relational language that will be considered the universal language. Also, the translation of the results back to a form usable by network and hierarchical databases will be quite a challenge. Added to this, work will have to take place for mapping of the data entities of network and hierarchical systems, and vice versa. It will take a number of thesis efforts before a heterogeneous DDBMS can be fully operational.

Final Comments

Future study should concentrate on the update

concurrency, the query optimization, the DDBMS reconfiguration, and the CNDD site software. Also, various translators will be needed to communicate between the various types of DBMSs. AFIT has a good start as far as hardware for this area by having LSI computers in the NETOS network, but will require much more powerful, multi-programming computers to handle both user and DDBMS software on the same system. Hopefully, years of research at AFIT will provide a greater understanding of the distributed database field.

Appendix A -- Glossary of Terms for DDBMS Design

Acknowledgements - Messages sent to another site in the network acknowledging the reception of some message from that network site.

Central Site - Site in the DDBMS that maintains the CNDD and pending update files, as well as performing regular DDBMS functions. Same as CNDD Site.

CNDD - Centralized Network Data Directory, contains locations of all data items in the DDBMS.

CNDD Site - Site in the DDBMS that maintains the CNDD and pending update files, as well as performing its regular DDBMS functions. Same as Central Site.

CNDD Site Failure - The system failure of the CNDD site, causing reproduction of the CNDD site data at another site in the DDBMS.

Command - A message that originates at a host computer, input by a user.

Data Locations - Numerical representations of sites in the DDBMS where data items are located.

DBMS - Database Management System, software module executing at host computers that organizes and retrieves data.

DDBMS - Distributed Database Management System, software modules executing with network protocol modules to combine databases together over network lines into a larger, single database.

Distributed Database - A configuration of database sites in

a network joined by DDBMS software into one, larger database.

ECNDD - Extended Centralized Network Data directory, a directory at every site in the DDBMS that contains a list of locations of data items that are updated from locations received from the CNDD.

External - Items entering the DDBMS software from outside sources, such as user or operator input.

File - Collection of data in an organized fashion either kept at a DDBMS site or sent between sites.

Host - The computer for a site which contains the local database and DBMS that interfaces with the DDBMS.

Host Queries - Queries that originate at the host computer that require only data from the host's database.

LNDD - Local Network Data Directory, a directory at every DDBMS site that lists data items that are located on the host's database.

Local - Items in the DDBMS relating to or originating at the host computer for a particular site.

Location - A numerical representation for a particular site in the DDBMS.

Message - A fixed-length, formatted piece of data that is transmitted from one site to another in the network.

Network - Software and communication lines that connect various host computers so that the computers can send and receive messages and files.

Network Malfunctions - Events that occur to sites and communications lines in a network that cause them not to function properly.

Network Queries - Queries originating at the host computer that have data at sites other than the host computer in the DDBMS.

Network-Determined - An event in the DDBMS that is handled by DDBMS software without user intervention.

Non-CNDD Site - A site in the DDBMS other than the CNDD site.

Output - Refers to messages and data sent to the user at a site that are the end products of actions taken by the DDBMS.

Pending Update - An update to data at a site in the DDBMS that has been accepted at a site where a copy of the data has occurred and is not yet transacted at an inactive site.

Pending Update File - A file of pending updates maintained at the CNDD site for any inactive site in the DDBMS.

Queries - Requests input by users for data at DDBMS sites that are written either in a local database language or the universal database language.

Query Results - Results of execution of a DDBMS query. The results can either be in the form of data which is the data on the host database that matches the parameters of the query, or a null file indicating that no data could be found to match the parameters.

Reconfiguration Requests - Messages in the DDBMS from users of the DDBMS or the DDBMS software that either request a change in the configuration of the DDBMS or notify sites that a change has already occurred.

Remote - Any data that originates at a site other than the site of reference.

Replicated Network Updates - Updates to data that is replicated at two or more sites in the DDBMS.

Request - A message that requires an action from a DDBMS site.

Site - A combination of the processor that executes DDBMS software and the host computer that are tied into the DDBMS. May be either one or two computers.

Site Crash - A DDBMS site becoming inactive due to a failure in the hardware at the site.

Status Information - A table at each DDBMS site indicating which sites are active and inactive in the DDBMS. Same as Status Information Table and Status Table.

Status Information Table - A table at each DDBMS site indicating which sites are active and inactive in the DDBMS. Same as Status Information and Status Table.

Status Table - A table at each DDBMS site indicating which sites are active and inactive in the DDBMS. Same as Status Information and Status Information Table.

Timeout - A software mechanism to wait for a return messaage from a site where a message has been sent. The length of

time to wait is arbitrary.

Translated - Converted from one database language or data model to another.

Unique Host Updates - Updates to data that exists only at the originating site's host computer.

Unique Network Updates - Updates to data that exists at a single site other than the originating site.

Update Results - Messages indicating either if an update is successful, or if there is some problem with the update.

Updates - Messages to update data in the DDBMS.

Appendix B

Requirements for the
Digital Engineering Laboratory
Distributed Database Management System

1. Initialize the DDBMS at startup time.
    a. Initialize if central site.
        1. Initialize data at central site.
        2. Query sites given in startup command for status.
        3. Evaluate responses from sites.
    b. Initialize if not central site.
        1. Initialize that site's data.
        2. Return the central site's query.
2. Maintain status information table on every site in DDBMS - table indicates whether each site is active or inactive.
3. Reconfigure DDBMS through the following actions:
    a. Add a site to DDBMS.
    b. Delete a site from DDBMS.
    c. Relocate the CNDD from one site to another.
    d. Recover from a DDBMS malfunction.
4. Transmit and receive messages and data between sites in the network, and between the network and the host computer.
5. Update and maintain network data directories.
    a. LNDD - Contains locations of data items for the site where it resides.
    b. CNDD - Contains locations for all data items in the

DDBMS. The CNDD exists only at the central site.

c. ECNDD - Exists at each site and contains locations of data remote to the site where it resides. The locations were previously retrieved from the CNDD.

6. Service incoming requests.

a. Service local queries and updates.

b. Service remote queries and updates.

c. If chosen the central site, service CNDD site requests.

1. Update the CNDD.

2. Retrieve locations from the CNDD.

3. Update pending update files.

Appendix C

Formats for Messages Transferred in the DDBMS

Description

This appendix contains three message formats. The first two formats are those messages which fit into the standard NETOS 32-character format, and the third format are those messages which do not fit into 32-character and are transferred in the DDBMS by way of files. The messages in each format are ordered alphabetically.

The following is a list of priorities of messages in the DDBMS:

1. Commands to bring up the DDBMS.
2. Commands to start execution and notify sites of a change in location of the CNDD.
3. Commands to rebuild the CNDD if the CNDD site crashes.
4. Commands to recover from a non-CNDD site crash.
5. Commands to add or delete a site from the DDBMS.
6. Updates to the LNDDs and the CNDD.
7. Commands to send a file and acknowledge an ability to receive a file.
8. Updates to database data and results of the updates.
9. Queries against database data and results of the queries.
10. Acknowledgements, ECNDD and pending updates.

Network Message

Char 0 - STX

1-3 - Three character message type, given below

4-6 - System ID at destination computer

7-9 - System ID at source computer

10 - Priority, given below

11-30 - Unused

31 - ETX

Message	Message Type	Priority
Added Site Message	ASM	5
Central Site Return Message	CSR	5
CNDD Established Message	CEM	3
CNDD Update Acknowledgements	CUA	10
Copy Finished Message	CFM	3
DDBMS Ready Command	DRC	2
Deleted Site Message	DSM	5
ECNDD Update Acknowledgements	EUA	10
File to Be Sent	FTS	7
Okay to Send File	FOK	7
Initial Central Site Contact Message	ICM	5
LNDD Request Message	LRM	3
Site N Startup Command	SNS	1
Site Status Query Message	SQM	1
Site Status Return Message	SRM	1

Host Message

Char 0 - STX

1-3 - Three character message type, given below

4-6 - System ID at site involved

7 - Priority, given below

8-30 - Unused

31 - ETX

Message	Message Type	Priority
External Add Site Command	EAS	5
External Delete Site Command	EDS	5
External Relocate CNDD Command	ERC	5

File Message

Char 0 - STX

1-3 - Three character message type, given below

4-6 - System ID at destination computer

7-9 - System ID at source computer

10 - Priority, given below

11-N - Data (N = length of the file)

Message	Message Type	Priority
Central Site Startup Command	CSS	1
CNDD Data Location Requests	CDL	7
CNDD Failure Notice to Operator	CFN	3
CNDD Update Messages to ECNDD	CUM	10
CNDD Updates	CUP	6
External LNDD Updates	ELU	6
External Recovery Command	ERC	4
Host Query Message	HQM	9
Local Queries	LQR	9
Local Replicated Network Update Message	LRN	8
Local Unique Network Update Message	LRU	8
LNDD Updates from CNDD	LUC	6
Output Replicated Network Update Results Message	ORN	8
Output Unique Network Update Results Message	OUN	8
Pending Update Requests	PUQ	10
Recovered DDBMS Message	RDM	3

Message	Message Type	Priority
Remote Query Message	RQM	9
Remote Query Requests	RQR	9
Replicated Network Update Messages	RNM	8
Replicated Network Update Results Message	RNR	8
Update Results from Host	URH	8
Updates to Network Data	UND	8
Updates to Unique Host Data	UUH	8
Unique Host Update Message	UHM	8
Unique Host Update Results Message	UHR	8
Unique Network Update Message	UNM	8
Unique Network Update Results Message	UNR	8

Appendix D

Retrieve Statement - Roth Relational Database System

Introduction

The Roth Relational Database System is a pedagogical tool for use by students at the Air Force Institute of Technology in studying database technology. The database system has several options including the System Options, the Define Options, the Edit Options, the Retrieve Options, and the Boss Option (10:111). Each of these options are discussed in detail in the Roth thesis (10).

The option of main concern to this thesis are the Retrieve Options (10:122-124). Inside the Retrieve Options section is a list of retrieve commands implemented in the Roth language. These commands are those produced at the user terminal by a program implemented as part of this thesis. All of the retrieve commands will be discussed, but note that only the PROJECT, SELECT, and JOIN operations are implemented in this thesis.

Union of Two Relations

The format is:

UNION relation1, relation2 GIVING relation3

where the first two relations must be union-compatible; that is, they have the same number of attributes and the ith attribute of one relation must be drawn from the same domain as the ith attribute of the other relation. Relation3 will

acquire the attribute names of relation1.

Example: UNION shippart, part GIVING upart

Intersection of Two Relations

The format is:

INTERSECT relation1, relation2 GIVING relation3

where all restrictions under UNION apply.

Example: INTERSECT shippart, part GIVING upart

Difference of Two Relations

The format is:

DIFFERENCE relation1, relation2, GIVING relation3

where relation3 = relation1 - relation2. All restrictions under UNION apply.

Cartesian Product of Two Relations

The format is:

PRODUCT relation1, relation2 GIVING relation3

where attribute names in relation3 will be the same as those in relation 1 and 2 except that duplicate names will be prefixed by the name of the relation it came from.

Example: PRODUCT shippart, part GIVING upart

Join of Two Relations

The format is:

JOIN relation1, relation2 WHERE attr1 op attr2 GIVING relation3

where attr1 is in relation1, attr2 is in relation2, and

op is =, <, or >. The JOIN operation is a subset of the cartesian product where the condition of membership is specified in the WHERE clause. All restrictions under PRODUCT apply.

Example: JOIN part, shipment WHERE part# = part# GIVING shipment-description

Project of a Relation Over a Subset of Its Attributes

The format is:

PROJECT relation1 OVER attr1, attr2, . . . attrN GIVING relation2

where attributes not specified in the OVER clause will be eliminated and any duplicate tuples will be eliminated.

Example: PROJECT shipment-description OVER color, supply# GIVING c-and-s

Select of a Subset of Tuples from a Relation

The format is:

SELECT ALL FROM relation1 WHERE condition GIVING relation2

where condition is a boolean prodicate on the attributes of relation1 of the form a1 AND/OR a2 AND/OR a3

op is =, <, or >. The expression may be fully parenthesized to indicate the proper precedence of the operators, but if not the AND has precedence over OR. One or more blanks or commas must be between each part of the command except that

the left parenthesis may be flush against an item to its right, and the right parenthesis may be flush against an item to its left.

Example: SELECT ALL FROM part WHERE location = Miami GIVING parts-in-Miami

Divide a Binary Relation by a Unary Relation

The format is:

DIVIDE relation1 BY relation2 OVER attr1 GIVING relation3

where relation1 is a binary relation, relation2 is a unary relation, and attr1 is an attribute of relation1 defined on the same domain as the attribute in relation2. Relation3 will be a unary relation with an attribute from relation1 that is not attr1.

Example: DIVIDE ps# BY s# OVER supply# GIVING p#

APPENDIX E

Publication Article

Report on

DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Introduction

A database is a large collection of data that is organized for rapid retrieval and updating. Software that controls data manipulation and structure in a database is known as a database management system (DBMS). Databases that reside on a single computer are known as centralized databases, while databases that are spread over several computers in a network are known as distributed databases. The software that manages tasks occurring in a distributed database is a distributed database management system (DDBMS).

A distributed database can provide a great enhancement to a multi-site organization, but at a significant cost in complexity. Users can access data on several computers, many more users can run transactions, and data integrity is maintained over several sites simultaneously. Serious problems to overcome in a distributed database include keeping data on each site concurrent, preventing data deadlock, and running queries and updates efficiently.

This paper will first cover the basic approaches to distributed databases and the interfaces of computers with a

DDBMS, and the data directories needed for a DDBMS. Following this will be a list of objectives set for the thesis project resulting in this paper. The objectives were for the requirements analysis, detailed design, and implementation phases of the DDBMS project, each of which will be described briefly. Finally, the results of the implementation and some conclusions will be discussed.

Approaches to Distributed Databases

There are three approaches to distributed database management systems. The integrated, homogeneous, and heterogeneous models are shown in Figure E-1 (5:7-10). In the integrated model, each DBMS is designed as being connected to the others in the network, and can access them without data translation, as shown in Figure E-1a. This strategy reduces the useful CPU time at each computer, and requires memory to store the data exchange process, two reasons for its lack of popularity.

The homogeneous model removes the network data exchange module from memory at each computer, and installs a communication software module between each computer and the rest of the network, as shown in Figure E-1b. In this model each computer must support the same DDBMS as the others. This is the most common method of implementing distributed databases.

The heterogeneous model can link different DBMSs

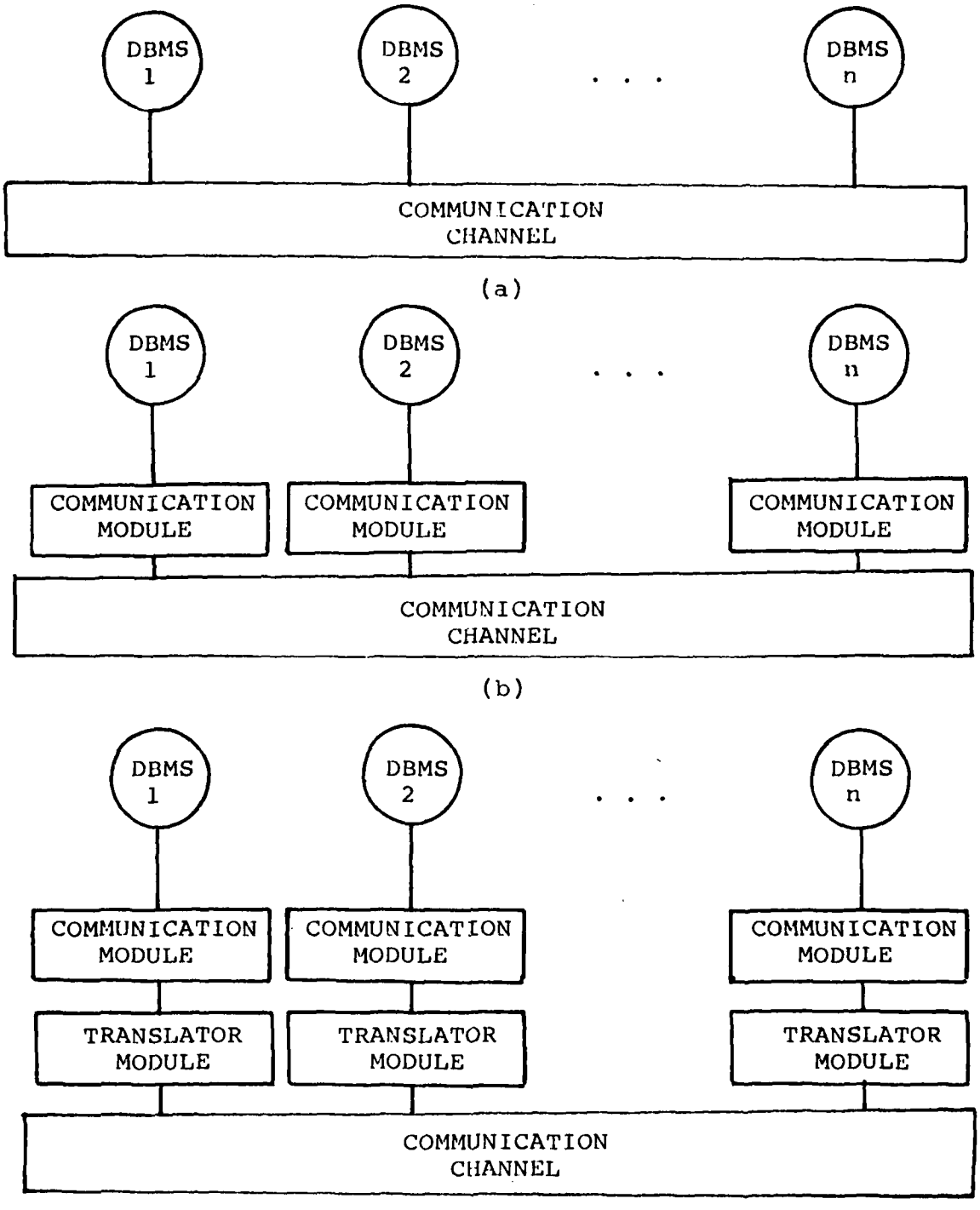


Figure E-1 (a) Integrated Architecture
(b) Homogeneous Architecture
(c) Heterogeneous Architecture (5:10)

together through a distributed database. Since the DBMSs are incompatible, a translator software module is installed between the communication module and the network communication channel, as in Figure E-1c. This is the most flexible model, but also the most complicated to implement. A universal database model would greatly enhance the usability of a heterogeneous model, since software could be written to translate from any type of DBMS language to the universal model, and from the universal back to the original language. Also, all network-wide functions in the DDBMS could be written in that one universal language.

Interfaces to the DDBMS

The basic interfaces to the DDBMS are the interfaces to the local DBMS, the network, the data directories, and the users of the DDBMS. The diagram for the interrelations of these interfaces is shown in Figure E-2. Some kind of translators are needed for the DDBMS version of queries and updates to be converted to those of the local DBMS language. An interface is needed directly to a user to give him the results of a DDBMS query or update. Interfaces to the network are needed to transmit messages from one DDBMS site to another concerning site status, queries, updates, voting, and other DDBMS transactions. Finally, directory interfaces are needed to allow the DDBMS software to search at the proper sites for required data.

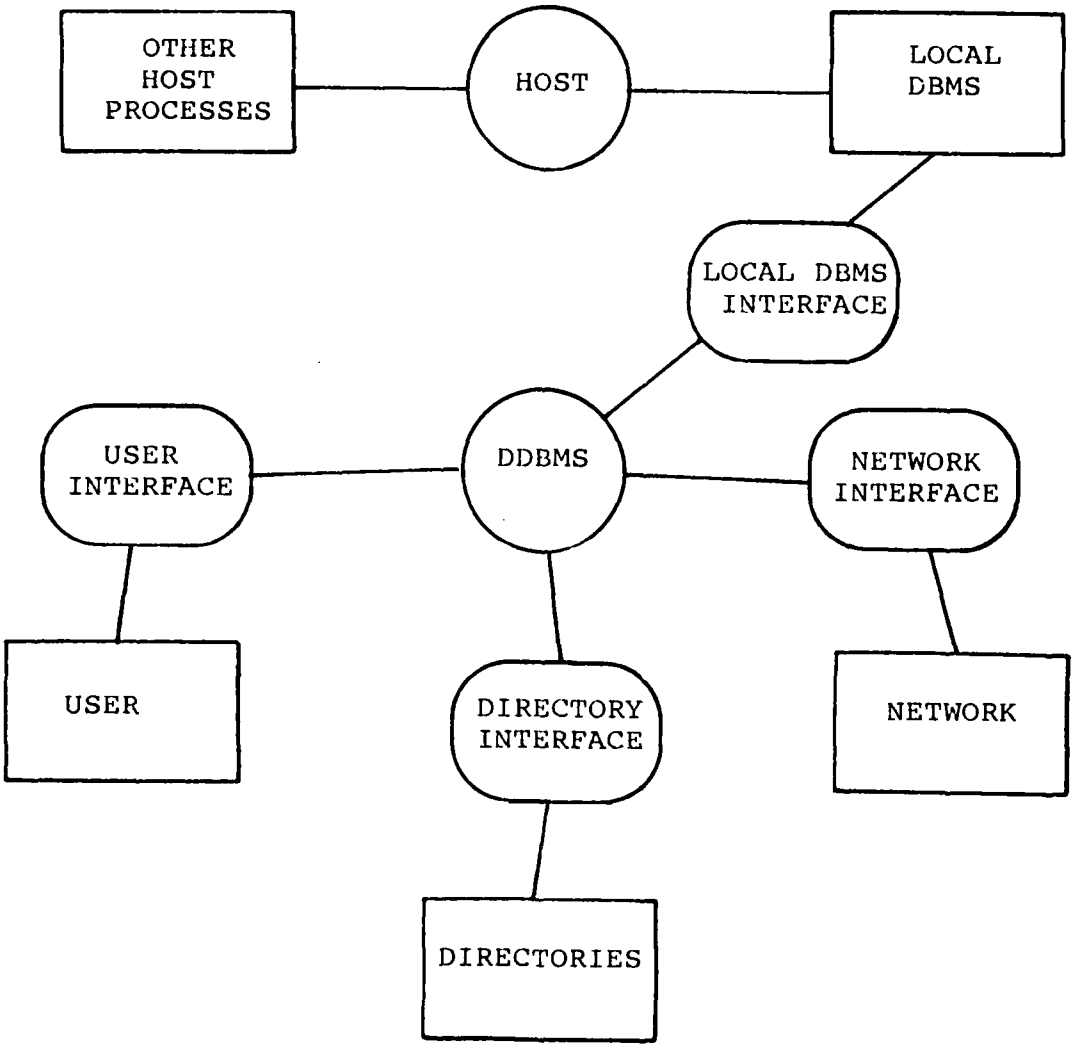


Figure E-2 Interfaces in the DDBMS

DDBMS Data Directories

The three data directories used in this design of a DDBMS are the Local Network Data Directory (LNDD), the Centralized Network Data Directory (CNDD), and the Extended Centralized Data directory (ECNDD). The LNDD is a directory which contains names of data entities that reside at the

DDBMS site where the LNDD is located. The CNDD contains locations for all data entities in the DDBMS. The ECNDD is a subset of the CNDD and contains CNDD entries of data entities used at a particular site whose locations were previously retrieved from the CNDD.

Objectives of the Thesis

This DDBMS research took place at the Air Force Institute of Technology (AFIT) to fulfill the thesis requirement for a master's degree. The research was intended to provide a foundation for future thesis projects at AFIT in distributed database research. This thesis project consisted of four basic phases: a DDBMS requirements analysis, a detailed design based on the requirements analysis, a partial implementation of the detailed design, and an analysis of the partial implementation.

Requirements Analysis

The required functions for the DDBMS fall into six basic categories:

(1) Initialize the DDBMS.

(2) Maintain status information on other sites.

(3) Reconfigure the DDBMS.

(4) Transmit and receive messages and data to other sites and to the host computer.

(5) Update and maintain the ECNDD and LNDD.

(6) Execute queries and updates. If chosen as the central site, executes CNDD and pending update functions.

These requirements were transcribed into SAD'T diagrams (9:62-64) to graphically display the analysis. The analysis was intended to be independent of any hardware, language, or specific algorithm.

The highest level function in the requirements is the module "Execute the DDBMS", shown in Figure E-3. It is decomposed into three modules. "Initialize DDBMS" prepares the individual sites in the DDBMS for execution at startup time. "Reconfigure DDBMS" changes the configuration of the DDBMS during execution by adding a site, deleting a site, moving the location of the CNDD site, or recovering from an unexpected malfunction in the DDBMS. "Execute DDBMS at Sites" processes all DDBMS site functions other than those accomplished by "Reconfigure DDBMS". These include processing incoming messages and producing a proper response as output.

An input message to a DDBMS site can be updates to that site's LNDD or ECNDD, or can be another type of message collectively called a request. This request can be one that originates at this local site, a local request, one that originates at another DDBMS site, a remote request, or one to this site if it happens to be the CNDD site, a CNDD site request. The routines to handles these requests is shown in Figure E-4.

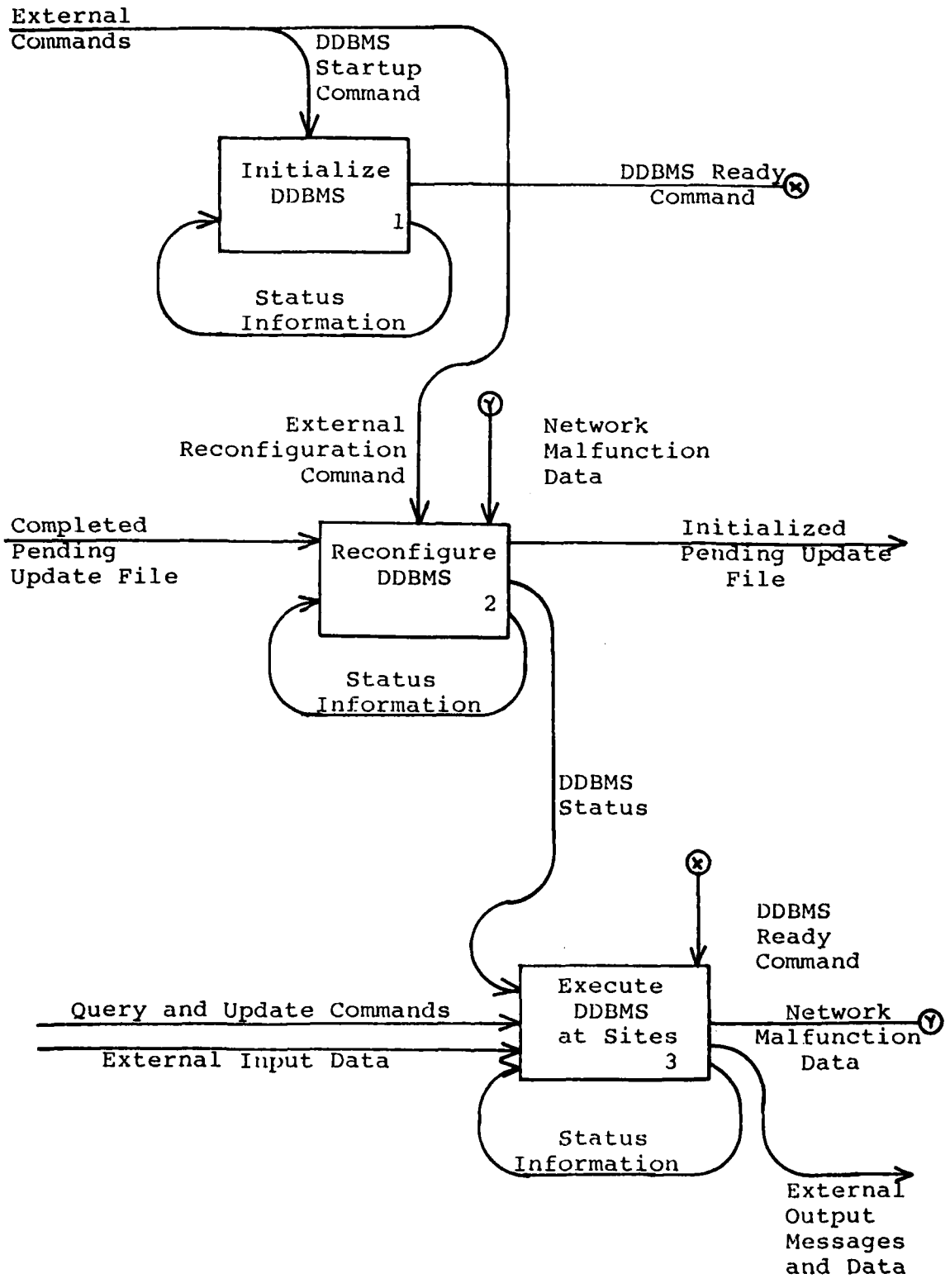


Figure E-3 SADT Activity "Execute the DDBMS"

"Service Local Requests" transacts query and update requests made at the local site. "Service Remote Requests" transacts query and updated requests made by other sites in the DDBMS to the local site. "Service CNDD Site Requests" handles CNDD data location requests, CNDD updates, and pending update requests, that are made to the site in the DDBMS where the CNDD is located.

Detailed Design

This phase of the project was a detailed design of a DDBMS based on the requirements analysis of the previous chapter. The idea of the design was to be as general as possible, but to limit the design in the specified areas according to the implementation decisions made. The generality comes from the fact that a large proportion of the design is applicable in most DDBMS configurations, and the implementation decisions were made to limit the amount of software design and code written for the query optimization and concurrency control of updates.

The highest level diagram of the structure charts is shown in Figure E-5 under the title "Execute DDBMS at Site N". In this design, Site N is referred to as the site where that particular software module is executing. There are three main functions for this module: initialization of the DDBMS at this site under "Initialize DDBMS at Site N", retrieving the next input message through "Get Next

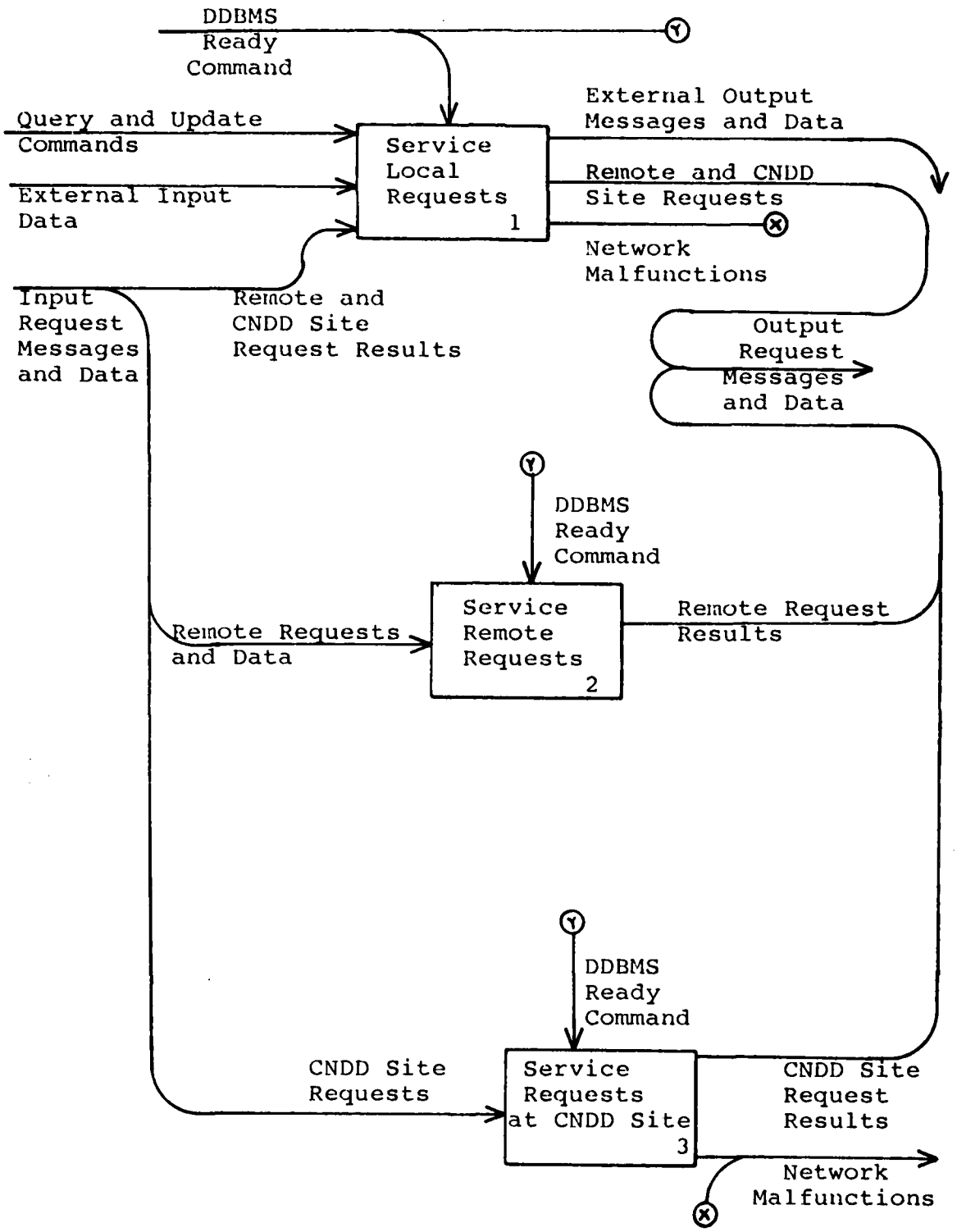


Figure E-4 SADT Activity "Service Requests"

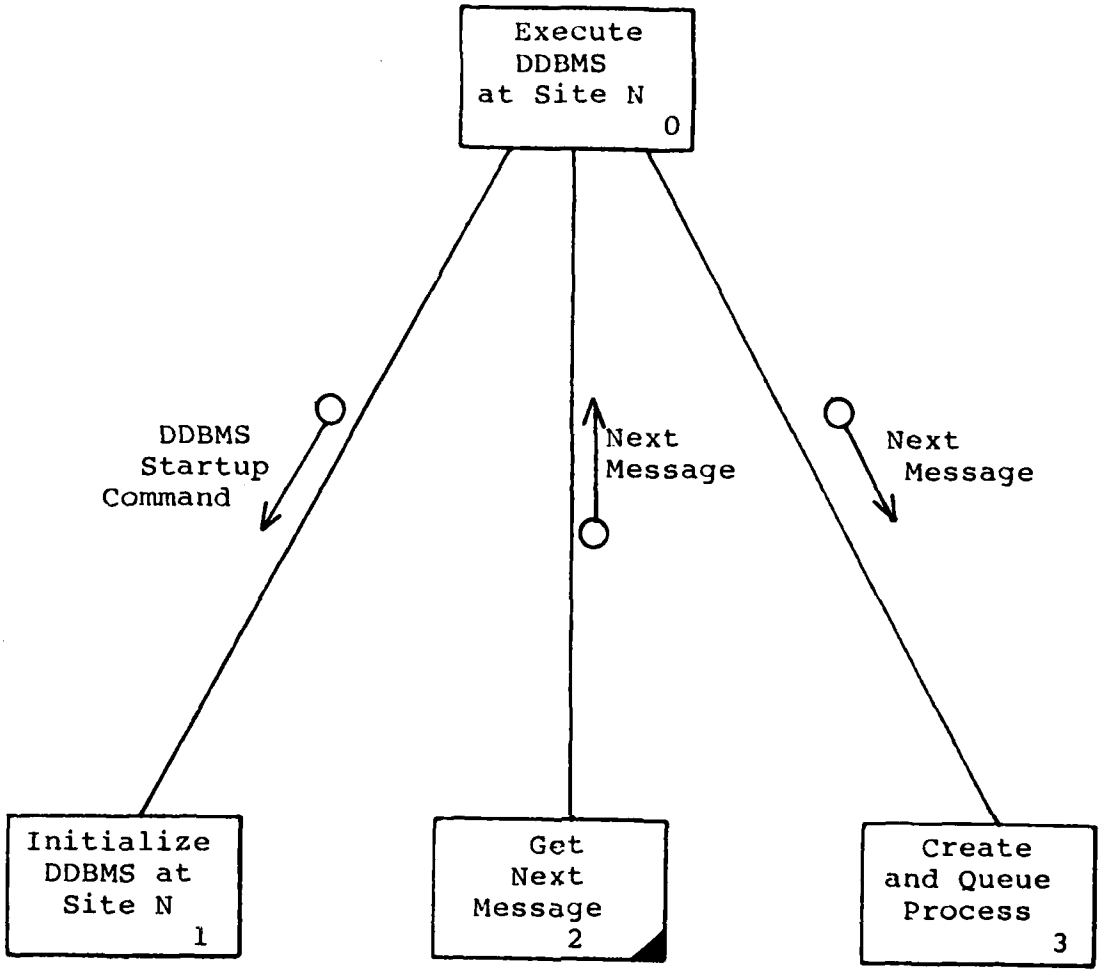


Figure E-5 Structure Chart for Process "Execute DDBMS at Site N"

Message", and creating a process for that message and placing it in the process queue under "Create and Queue Process".

Processes in the DDBMS queued under "Create and Queue Process" are released for execution in the DDBMS software by

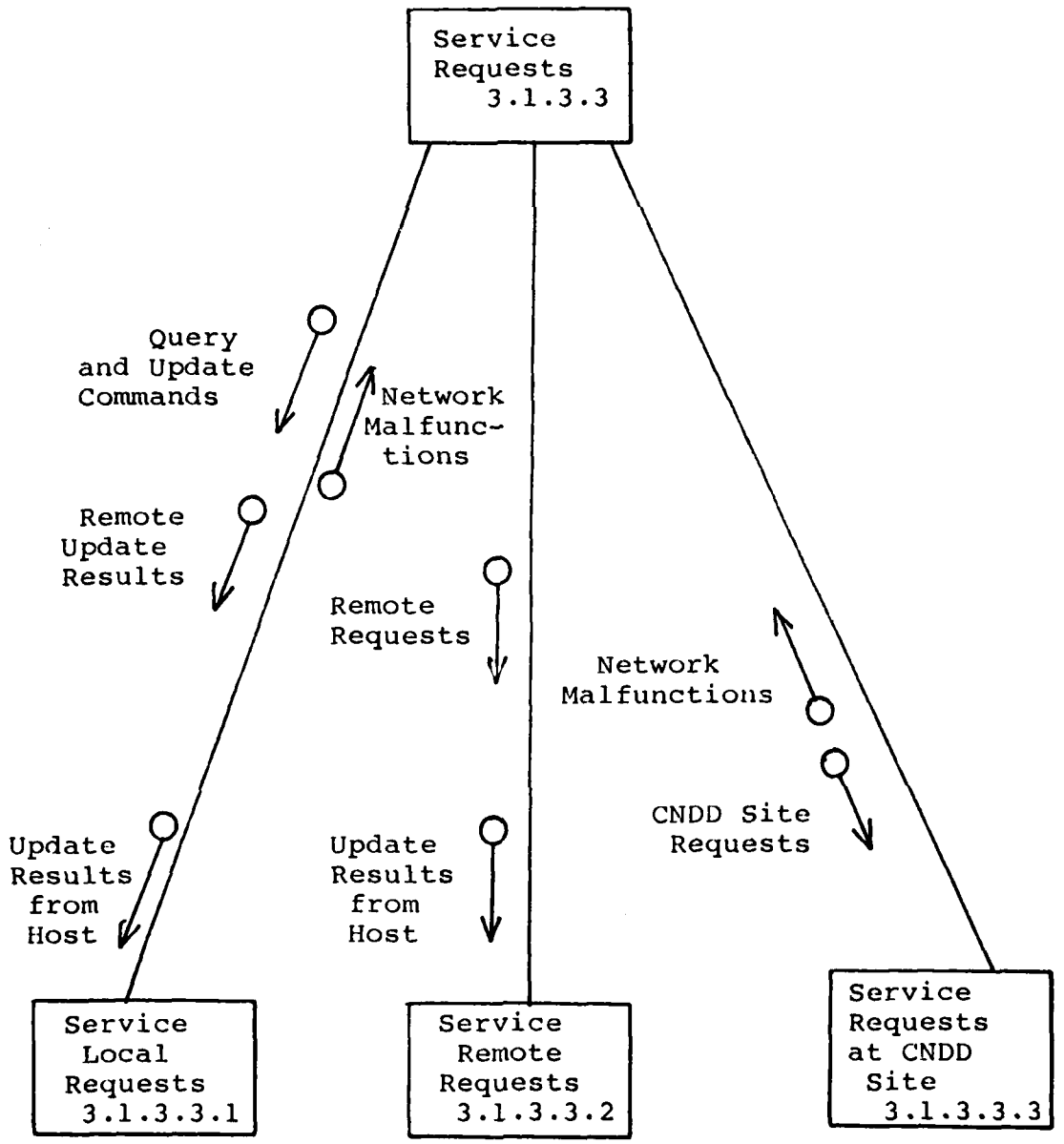


Figure E-6 Structure Chart for Process "Service Requests"

a system scheduler. The messages in these processes can be

requests to reconfigure the DDBMS and other data transaction requests. Each of these are handled by separate routines. The routine to handle data transaction requests is shown in Figure E-6. As in Figure E-4 of the Requirements Analysis, the routine is decomposed into "Service Local Requests", "Service Remote Requests", and "Service Requests at CNDD Site". These routines handle, as in Figure E-4, locally-originating updates and queries, remotely-originating updates and queries, and CNDD site requests, respectively.

Partial Implementation

A part of the DDBMS structure chart design discussed in the previous chapter was implemented. Two LSI-11 microcomputers in the LSINET of AFIT were chosen as sites to hold the DDBMS software that would obtain query data from an S-100 computer executing the dBASE II relational DBMS and a VAX 11/780 running the UNIX operating system with INGRES, also a relational DBMS. A relational DBMS language known at AFIT as Roth's Relational DBMS was used for all input queries. No data updates of any kind can be sent over the network under this implementation.

Figure E-7 shows a pictoral representation of the architecture to implement the DDBMS architecture. The computer architecture chosen for this partial implementation consists of an S-100 microcomputer running dBASE II; a VAX 11/780 running the UNIX Operating System and the relational DBMS INGRES; two LSI-11 microcomputers (System A and System

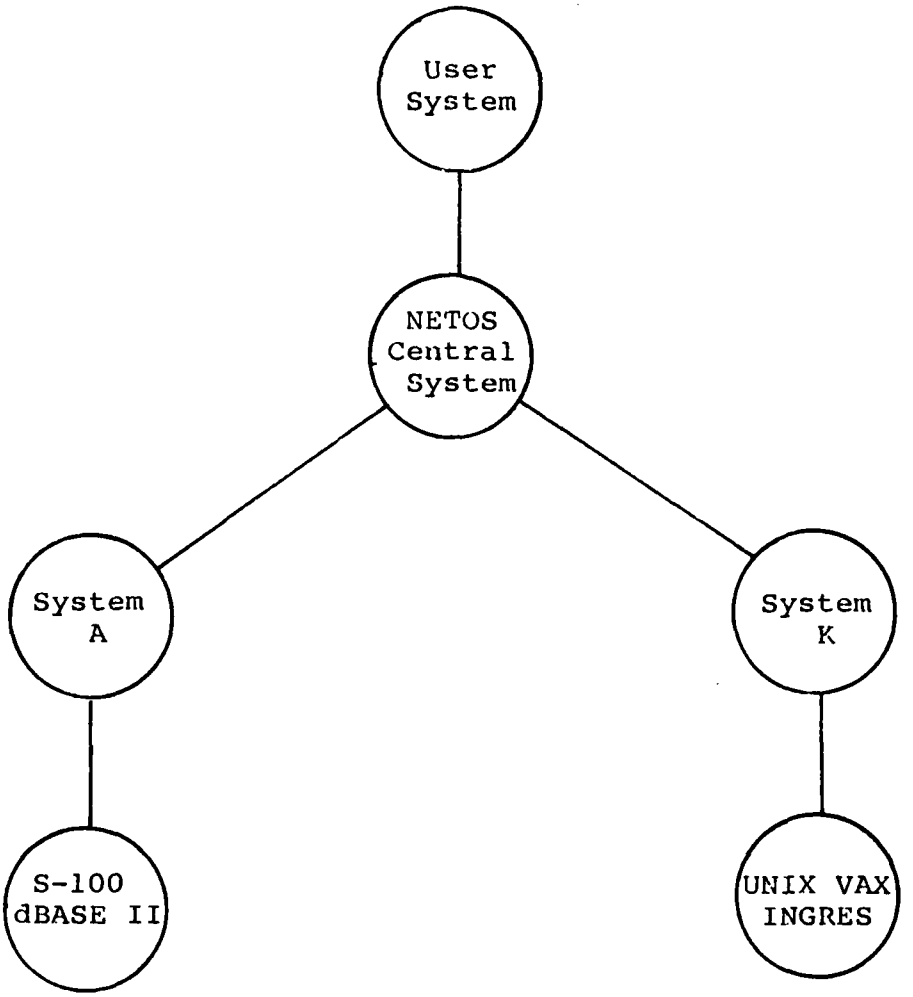


Figure E-7 DDBMS Implementation Architecture

K) connected in a network via the Network Operating System (NETOS) (4:1-19) where System A is connected to the S-100 and System K is connected to the VAX; a third LSI-11 computer, System B, which acts as the central system in NETOS through which all network messages are passed; and a final "user" LSI-11 computer where queries originate and where the results appear.

Only transfer and execution of queries has been accomplished in this implementation. System A and System K will have identical software except for the commands sent to the respective dBASE II and INGRES systems to translate and execute a query and the software to handle multiple-site queries. The systems will contain algorithms to update and retrieve data only from the ECNDD and the LNDD, and the ECNDD is assumed to have all network location data. All input queries originate at the user system and are formatted in Roth's Relational DBMS language (10:122-124) by a program that prompts the user for relation names, attribute names, and conditions. Queries are sent to either the S-100 computer with dBASE II, or the VAX computer with INGRES. System A and System K are connected via the central system in NETOS, and send queries to each other to retrieve data from the host computer (the S-100 or the VAX 11/780) of the other site.

There are two main translation modules involved in this partial implementation of DDBMS software. The first module is one that translates Roth's Relational DBMS queries into dBASE II command files that can execute on the S-100 computer. This module was written by Gunning in December 1983 as a part of an introductory database course at the Air Force Institute of Technology (3:2-3). The second module is a version of the dBASE II translator modified as part of this thesis effort to be used by INGRES under UNIX running

on the VAX 11/780 computer. Since the Gunning translator could only handle queries and not updates, it was decided early in the implementation phase to direct the main effort toward transferring only queries and query results by the DDBMS software.

Due to the large nature of the DDBMS design and the small amount of time with which to accomplish this partial implementation, only a small portion of the actual structure chart design was implemented. The main idea was to limit the implementation to only queries and query results being passed over the network in the DDBMS. The LNDD and the ECNDD were implemented as flat files updated by a screen editor. The LNDD contains only data names and associated databases located at the host site, and the ECNDD contains relation names, database names, and network site locations. The ECNDD is assumed to have data on all relations in the DDBMS.

All mechanisms to recover from DDBMS malfunctions were left out of this implementation along with all maintenance of pending update files. Also, all data updates were left out of this implementation due to the lack of an effective update translator and the inherent difficulty in keeping updates occurring concurrently.

The final implementation could effectively create queries at the user site, process them at the DDBMS sites, System A and System K, execute them on the host computers,

the VAX and the S-100 computer, and return the results to the user site. The most difficult part implemented was that of a multiple-site JOIN, where one relation of the JOIN appeared at one DDBMS site and the other relation at the other site. The next section will discuss results of queries testing the DDBMS.

Testing and Evaluation

The software written during the implementation phase was tested to see its performance accuracy. Cases were devised to test both the quality of the data that was output and the speed of execution. Only the software written for this thesis was tested thoroughly, though modules such as the Roth-dBASE II translator and the NETOS ISO Layers had to perform successfully for the implementation to work.

In the production of the DDBMS software running the LSI computers in the Digital Engineering Laboratory, each module was again individually tested and connected together for a system test. For this testing, relations were chosen on the S-100 and VAX systems. The list of queries used in the testing is in Figure E-8. Note that SELECT, PROJECT, and JOIN queries were chosen so that one of each could be done with data residing at a particular site, and also a multiple-site JOIN is executed. Each of these queries was sent to both System A and System K. The results were evaluated to be correct.

a) JOIN supply, parts WHERE pnum = pnum GIVING newrel
(one site INGRES JOIN)

b) JOIN supply, shipment WHERE snum = snum GIVING newrel
(multiple-site JOIN)

c) JOIN bolts, nuts WHERE type = type GIVING newrel
(one site dBASE II JOIN)

d) SELECT ALL FROM parts WHERE ((color = black) or (color = gray)) and weight > 500 GIVING newrel
(INGRES SELECT)

e) SELECT ALL FROM bolts WHERE (pnum > 30) and (quan > 400) GIVING newrel
(dBASE II SELECT)

f) PROJECT supply OVER snum, pnum, jnum GIVING newrel
(INGRES PROJECT)

g) PROJECT nuts OVER pname, type GIVING newrel
(dBASE II PROJECT)

Figure E-8 List of Queries Used in DDBMS Testing

Figure E-9 is a graph of the execution times of the queries from Figure E-8 run in the DDBMS. Note the relative slowness of the queries that required data from the UNIX VAX (frequently overloaded) compared to those that did not require UNIX VAX data. Also note that queries requiring data from more than one computer ran longer than those which required data from only one computer. Finally, note that queries requiring data from the other computer ran longer than those just needing data from that site's host computer.

Conclusions

There are several projects that can be done at AFIT to follow after this thesis effort. First, an update

Figure E-8

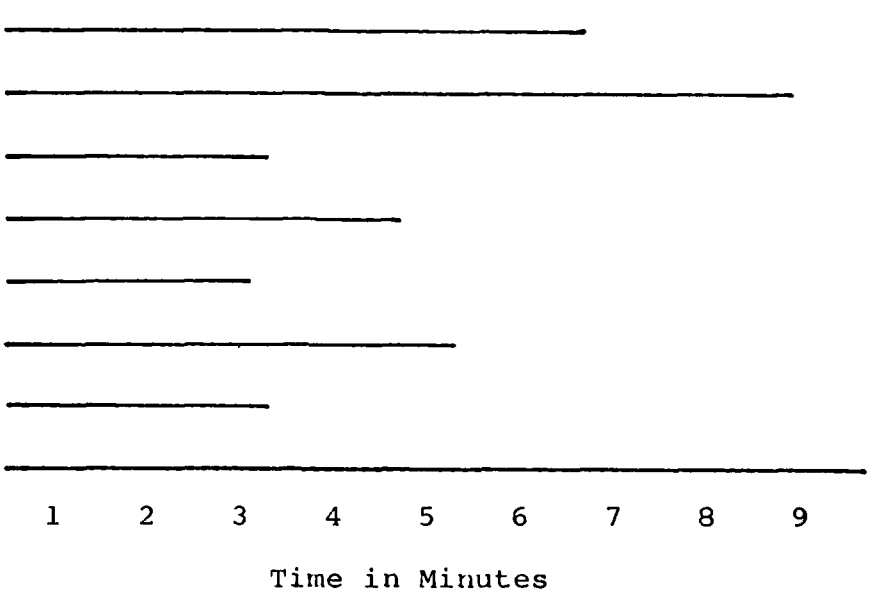


Figure E-9 Graph of Total Execution Times for DDBMS Queries

translator is needed in the DEL. Second, the functions of the CNDD and maintaining pending updates need to be implemented. Third, DDBMS update software should be implemented to include an update concurrency algorithm. Fourth, a DDBMS query optimization algorithm is required to direct query parts to optimum sites in the DDBMS. Fifth, software should be implemented to reconfigure the DDBMS when needed and in case of possible catastrophe. Sixth, more

sophisticated translators are essential to handling more complicated relational DBMS statements and in the conversion of statements from a network or hierarchical DBMS language to a relational one. Seventh, algorithms should be implemented to handle input, process, and output queues to transfer messages between DDBMS sites. Finally, once the above projects are finished, the system needs to be converted to be heterogeneous, that is, to handle multiple types of DBMS languages as inputs.

This thesis effort barely scratched the surface of a huge and widely expanding distributed database research area. It provided a design basis for the projects mentioned earlier in this chapter to continue research at AFIT in this area. The field is very wide and demanding, too large to cover completely in one thesis.

The major areas of study should be the update concurrency, the query optimization, the DDBMS reconfiguration, and the CNDD site software. Also, various translators will be needed to communicate between the various types of DBMSs. AFIT has a good start as far as hardware for this area by having LSI computers in the NETOS network, but will require much more powerful computers to handle the load of DDBMS software. Especially helpful would be multi-programming computers that could run both DDBMS software as well as user/DBMS application software concurrently on the same system. Hopefully, years of

research at AFIT will provide a greater understanding of the distributed database field.

Bibliography

1. Coles, R. J. et al. "WIS Joint Mession Applications--Database Management and Distributed Processing Recommendations," Mitre Corporation, Bedford, MA, January 1984 (AD-A090 025).

2. Dawson, Jeffery L. "A Transaction Workload Model and its Application to a Tactical C3 Distributed Database System," Mitre Corporation, Bedford, MA, July 1980 (AD-B080 706L).

3. Gunning, Chris R. "A Roth Relational DBMS to dBASE II Query Translator," Project for Introduction to Database Systems, EE6.46, December 1983.

4. Hartrum, Thomas C. "LSINET Network Protocol, ISO Model," Version 1.0, Documentation Paper, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, July 1984.

5. Imker, Capt Eric F. Design of a Distributed Database Management System for Use in the AFIT Digital Engineering Laboratory, MS Thesis GCS/EE/82D-21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1982.

6. Lin, W. K. et al. "Distributed Database Control and Allocation. Volume 1. Framework for Understanding Concurrency Control and Recovery Algorithms." Computer Corporation of America, Cambridge, MA, October 1983 (AD-A138 891).

7. -----. "Distributed Database Control and Allocation.

Volume 2. Performance Analysis of Concurrency Control Algorithms." Computer Corporation of America, Cambridge, MA, October 1983 (AD-A138 892).

8. -----. "Distributed Database Control and Allocation. Volume 3. Distributed Database System Designer's Handbook." Computer Corporation of America, Cambridge, MA, October 1983 (AD-A138 893).

9. Peters, Lawrence J. Software Design: Methods and Techniques. New York: Yourdon Press, 1981.

10. Roth, Mark A. The Design and Implementation of a Relational Database System, MS Thesis, GCS/EE/79-14. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1979.

11. Tanenabaum, Andrew S. "Network Protocols," Computing Surveys, 13: 453-487 (December 1981).

12. Thomas, Robert H. "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases," Tutorial: Distributed Data Base Management, 88-94. New York: IEEE Computer Society, 1978.

13. Tsuchiya, M. and Mariani, M. P. "Distributed Database Performance Modeling," TRW Incorporated, Colorado Springs, CO, September 1983 (AD-A090 025).

14. Wong, Eugene. "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," Tutorial: Distributed Data Base Management, 50-68. New York: IEEE Computer Society, 1978.

15. Woodfill, John et al. INGRES Version 6.3 Reference Manual, University of California at Berkeley, Berkeley, California, April 1981.

VITA

Captain John G. Boeckman was born on 5 December 1955 in Omaha, Nebraska. He graduated from high school in Omaha in 1974 and attended the University of Nebraska-Lincoln from which he received the degree of Bachelor of Science in Mathematics in May 1978. In October 1978, he entered Officer Training School at the Lackland Training Annex, Texas, where he received his commission in the USAF in January 1979. He attended the Computer Systems Analyst Course at Keesler AFB, Mississippi until April 1979, when he was assigned to the 3900 Computer Services Squadron, Offutt AFB, Nebraska. He was assigned there until entering the School of Engineering, Air Force Institute of Technology in May 1983.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/84D-5	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Ctr; 8b. OFFICE SYMBOL (If applicable): COTD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Griffis AFB, New York 13441	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): John G. Boeckman, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1984 December; 15. PAGE COUNT: 139	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Distributed Database, Network, Database, Distributed Database Management System
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: DESIGN AND IMPLEMENTATION OF THE DIGITAL ENGINEERING LABORATORY DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Thesis Chairman: Dr. Thomas C. Hartrum

Approved for public release: IAW AFR 190-17
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433
21 Feb 85

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED X ☐ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Thomas C. Hartrum	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576; 22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This effort produced a basic design and partial implementation of a distributed database management system (DDBMS) for use in the AFIT Digital Engineering Laboratory. The objectives of this thesis were to lay out the requirements for a DDBMS, to design a simplified implementation of one, and to accomplish a partial implementation of that design.

The requirements analysis used the Structured Analysis and Design Technique (SADT) to document the DDBMS requirements. The analysis, independent of any hardware or specific algorithmic implementation, covers all aspects of a DDBMS, including routing and execution of queries and updates, DDBMS initialization and reconfiguration, and recovery from network malfunctions.

The detailed design expanded the SADT diagrams of the requirements analysis for specific methods of executing queries and updates in the DDBMS. These methods were selected to limit the scope of that design. Structure charts were produced using the SADT diagrams as a reference, specifying parameters and algorithms for the modules.

Only the part of the design that handles DDBMS queries was implemented. The implementation was greatly simplified to exclude query partitioning and optimization. Two DDBMS nodes were connected to host computers that evaluate queries and return the resulting relation.

END

FILMED

5-85

DTIC